FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Primer on Abstract Interpretation

VCLA, Winter School on Verification, 6-10 February 2012, Vienna

Florian Zuleger, TU Vienna

Static Program Analysis

Aims at proving runtime properties of programs without actually executing the program.

Examples:

- Is the program free of runtime errors, e.g. overflows, division by zero?
- Are two structures in the heap disjoint?
- Does the program terminate?

Importance of Static Analysis

Traditionally static analysis is used in optimizing compilers that transform programs in ways that preserve the program behavior.

Today, static analysis is also applied for

- program understanding
- profiling
- testing
- **verification**

Problem Statement

```
x = y = 5
while (*)
   x++; y++;
while (x > 1)
   x--; y--;
assert(y == 1);
```

Can the assertion be violated?

Problem Statement

```
x = y = 5
while (*)
   x++; y++;
while (x > 1)
   x--; y--;
assert(y == 1);
```

Can the assertion be violated?

The star * represents non-deterministic choice.
It is used for conditions that we do not want to or cannot model, e.g., string1[x] = string2[y]

Problem Statement

```
x = y = 5
while (*)
    x++; y++;
while (x > 1)
    x--; y--;
assert(y == 1);
```

Can the assertion be violated?

We are interested in the reachable states at the assertion!

Problem Statement

```
x = y = 5
while (*)
   x++; y++;
while (x > 1)
   x--; y--;
assert(y == 1);
```

Can the assertion be violated?

We are interested in the reachable states at the assertion!

Classical dataflow analyses are too imprecise!

Reachable States

Formally, the reachable states can be obtained as the least solution to a fixed point equation (collecting semantics).

However, the reachable states are not computable in general:

- infinitely many states (consider the values of x and y)
- non-terminating computation (the first loop)

Refined Problem Statement

Thus, we are interested in analyses that are

- **sound**

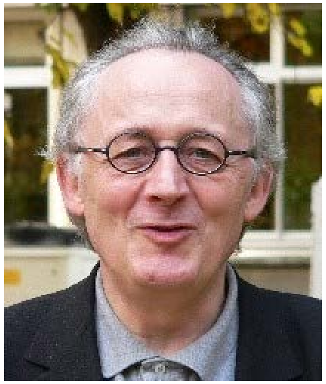
  $\Rightarrow$ The computed result of the analysis is a superset of the reachable states.
- **effectively computable**

  $\Rightarrow$ The analysis converges in finite time.
- **sufficiently precise in practice**

  $\Rightarrow$ Depends on the programs in the problem domain.

Abstract Interpretation
(Cousot, Cousot, POPL'77)

Abstract interpretation offers a systematic way for designing static analyses:



- Abstract elements denote a (possibly infinite) set of concrete program states.
- The concrete state transformers (i.e, the program statements) are replaced by abstract state transformers.

Abstract Interpretation

- Abstract interpretation computes a fixed point in the abstract.
- The computation is done on abstract elements using the abstract state transfomers.
- The result of the analysis is always correct, but may be imprecise.
- Acceleration techniques enforce the termination of the analysis.

Outline

1. Introduction
2. Zone Abstract Domain
3. Correctness of Abstract Interpretation
4. Overview on Abstract Domains
5. Discussion of Abstract Interpretation

What abstraction do we need?

```
x = y = 5
while (*)
   x++; y++;
while (x > 1)
   x--; y--;
assert(y == 1);
```

Abstract variable values by
signs, e.g., x is positiv?
intervals, e.g., $x \in [5,10]$?

Relational Abstract Domains

```
x = y = 5
while (*)
   x++; y++;
while (x > 1)
   x--; y--;
assert(y == 1);
```

Abstract variable values by signs, e.g., x is positiv? intervals, e.g., $x \in [5,10]$?

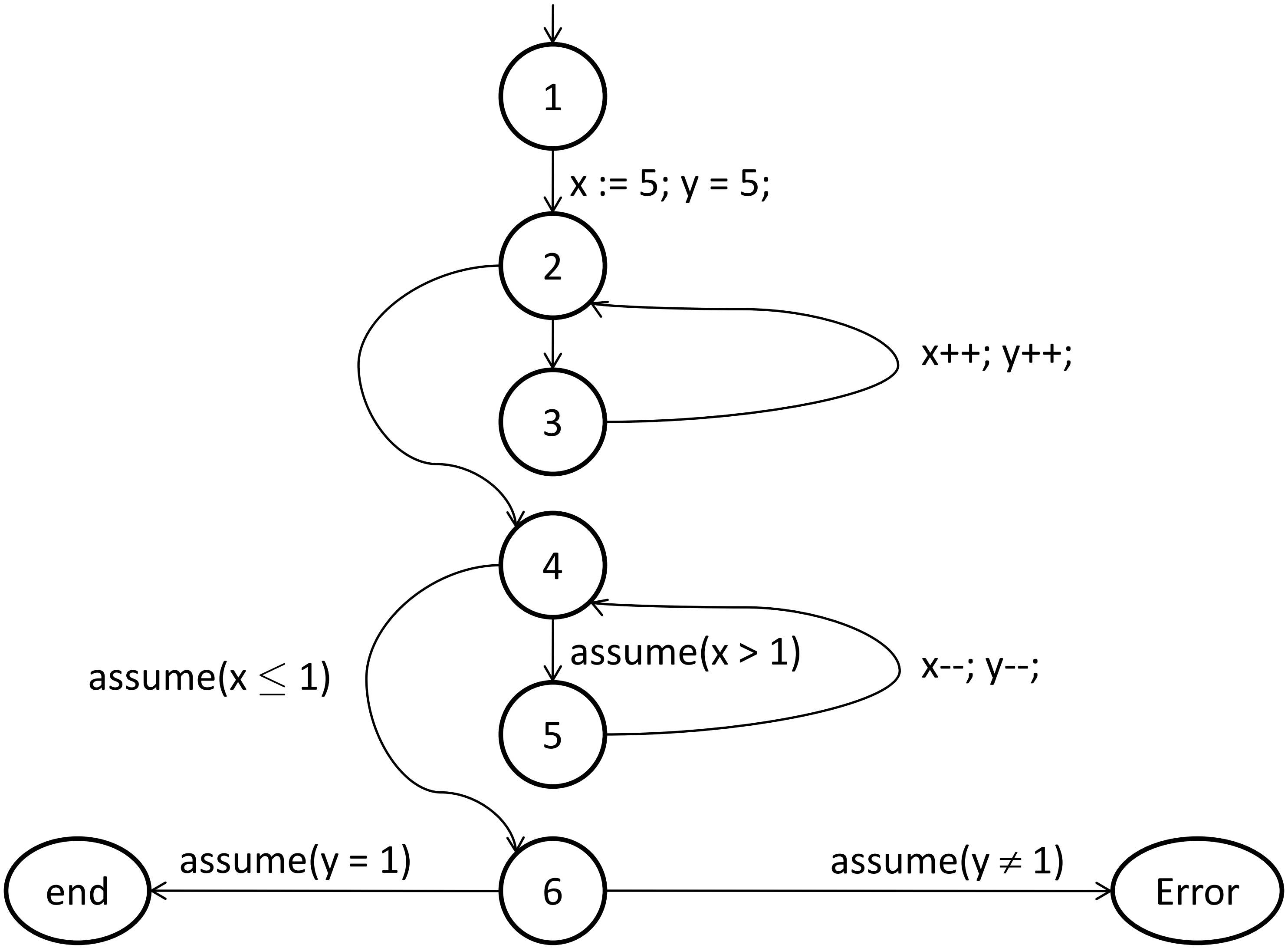
We need a relational abstract domain! Key property: x=y

Labeled Transition System

A **Labeled Transition System (LTS)** consists of

- a (finite) set L of **program locations,**
- a set $E \subseteq L \times A \times L$ of **edges**, where the set of **labels** A consists of the following statements:
  - – assign statements *id = E;*
  - – assume statements *assume(E);*

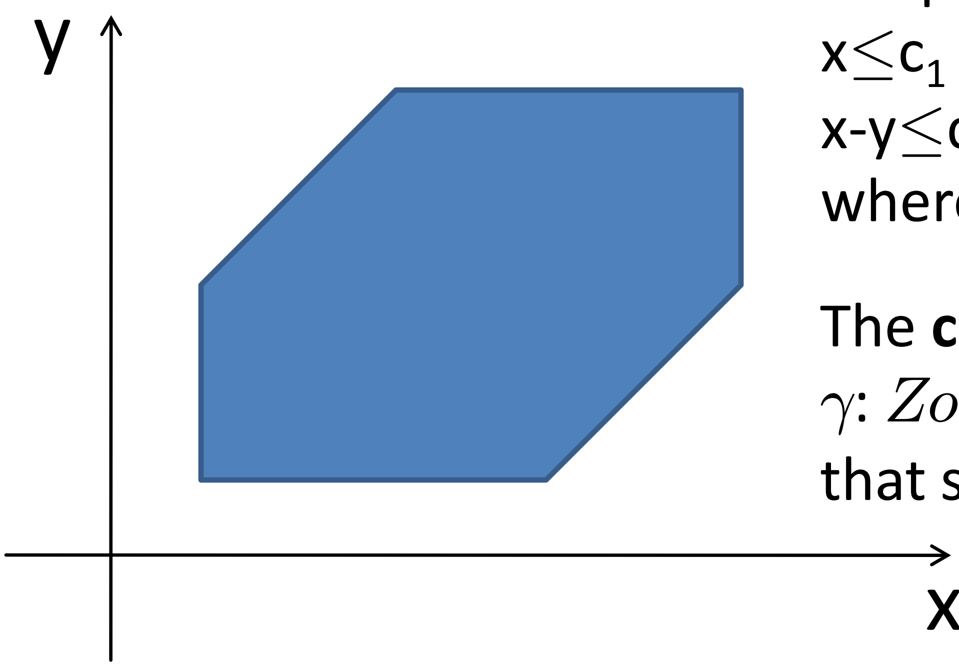
Example



1
x := 5; y = 5;
2
x++; y++;
3
4
assume(x ≤ 1)
assume(x > 1)
x--; y--;
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

The Zone Abstract Domain

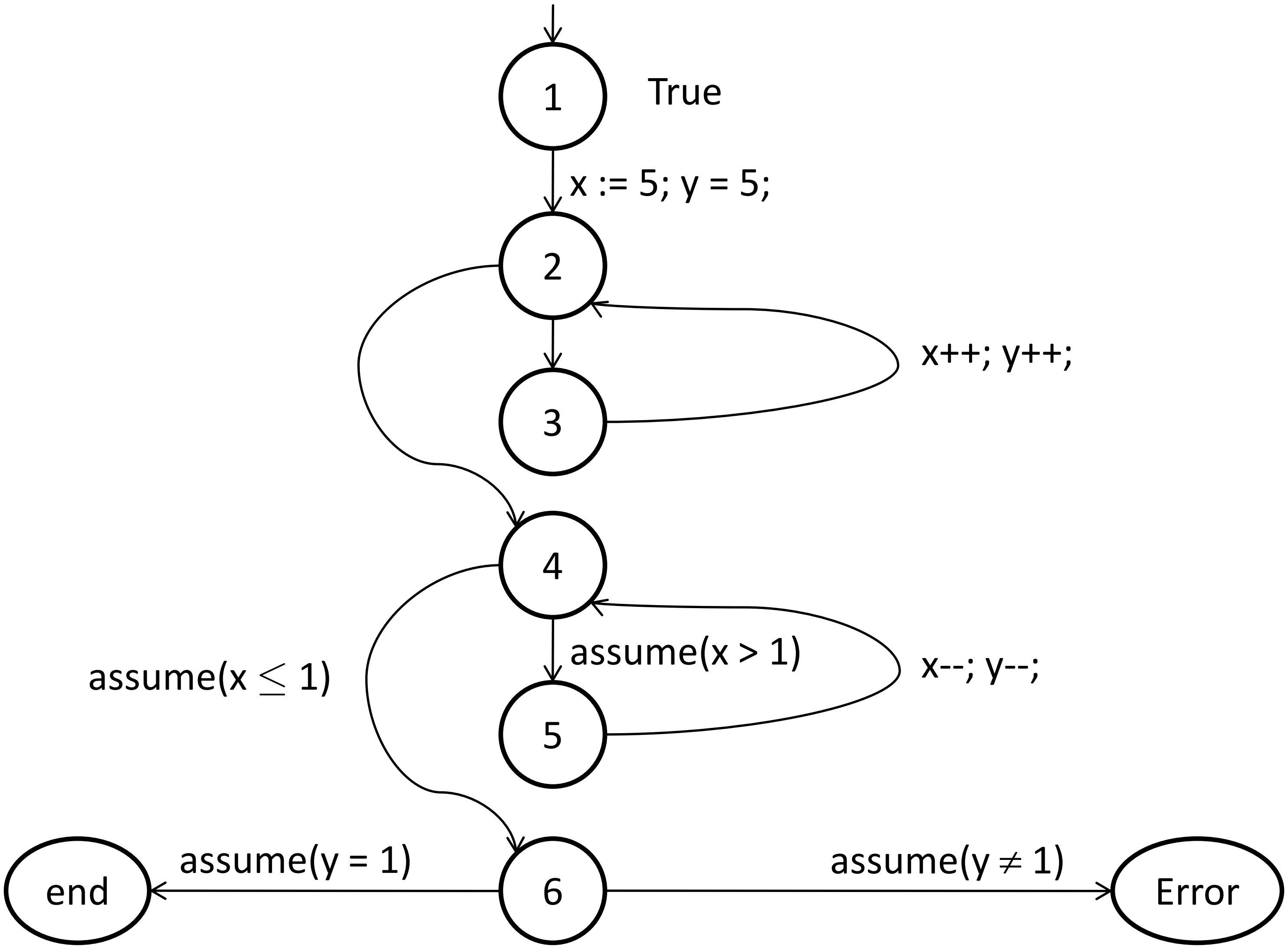
The zone abstract domain describes sets of values through certain polygons as pictured below.



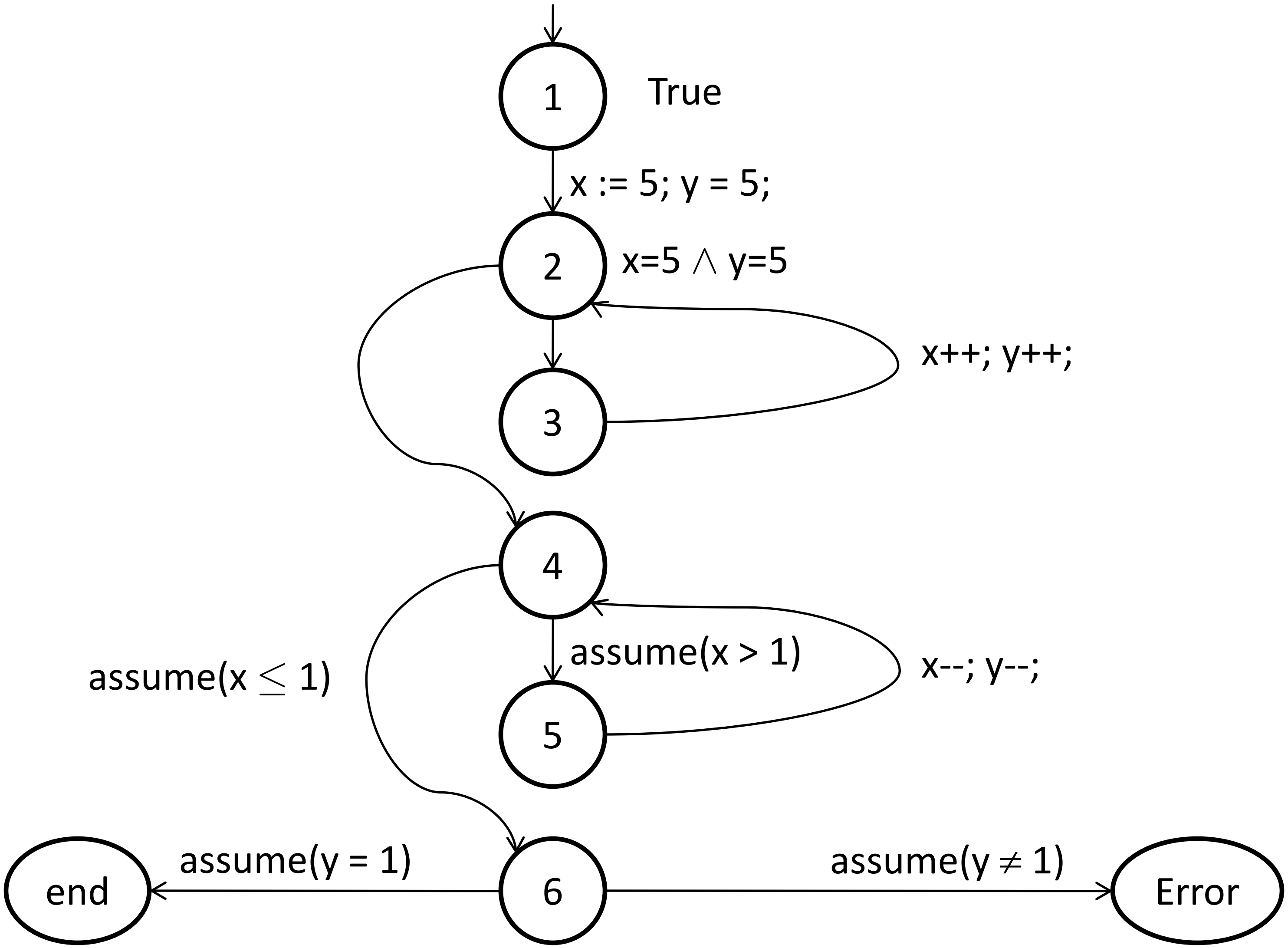
Formally, zones consist of linear inequalities:
$x \leq c_1 \wedge -x \leq c_2 \wedge y \leq c_3 \wedge -y \leq c_4 \wedge x-y \leq c_5 \wedge y-x \leq c_6$,
where $c_1,\ldots,c_6 \in \mathbb{Z} \cup \{\infty\}$

The **concretization function** γ: $Zone \rightarrow \mathbf{2}^{Val}$ returns all values that satisfy the inequalities.

Example

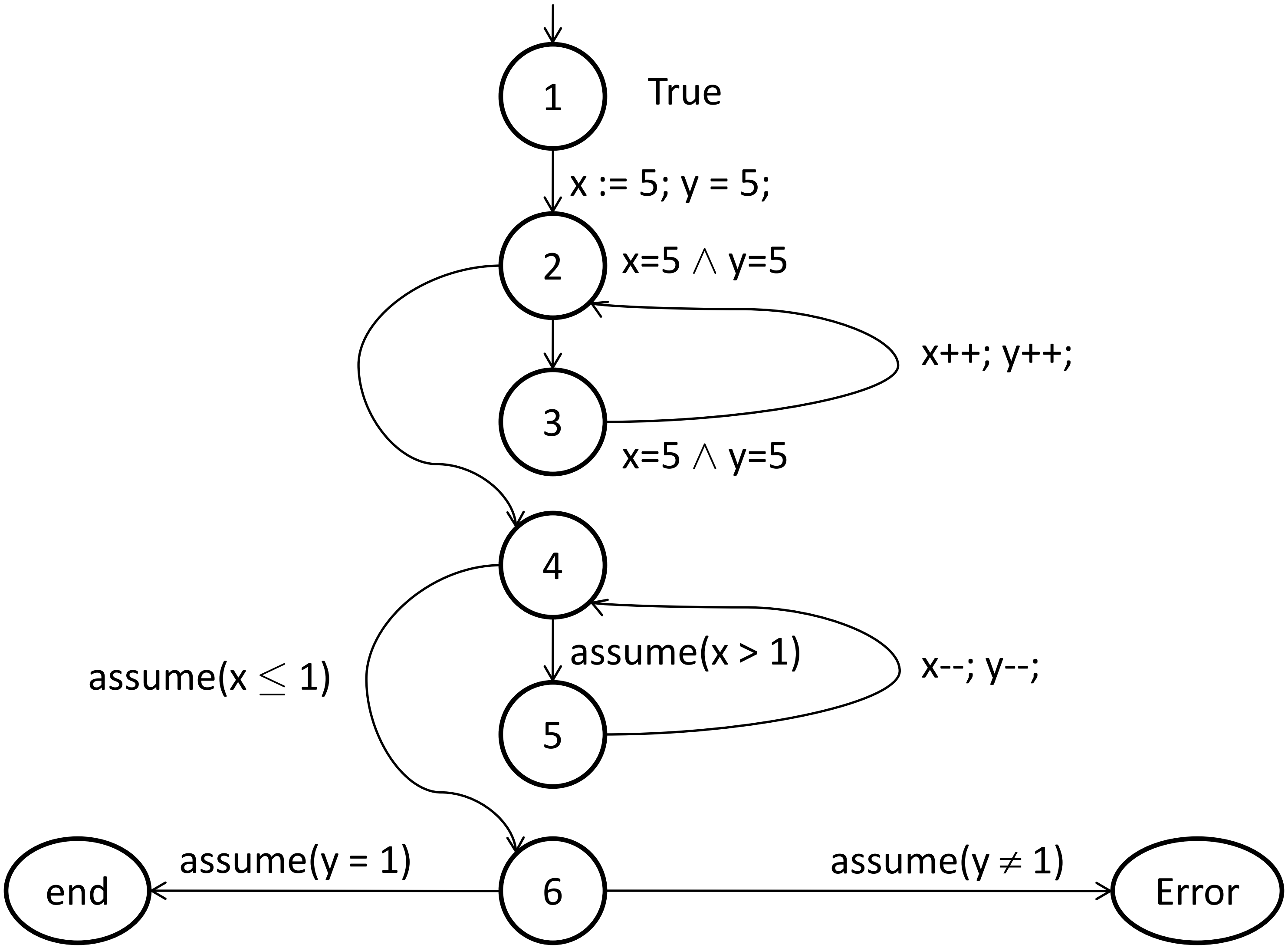

1
True
x := 5; y = 5;
2
x++; y++;
3
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

post#

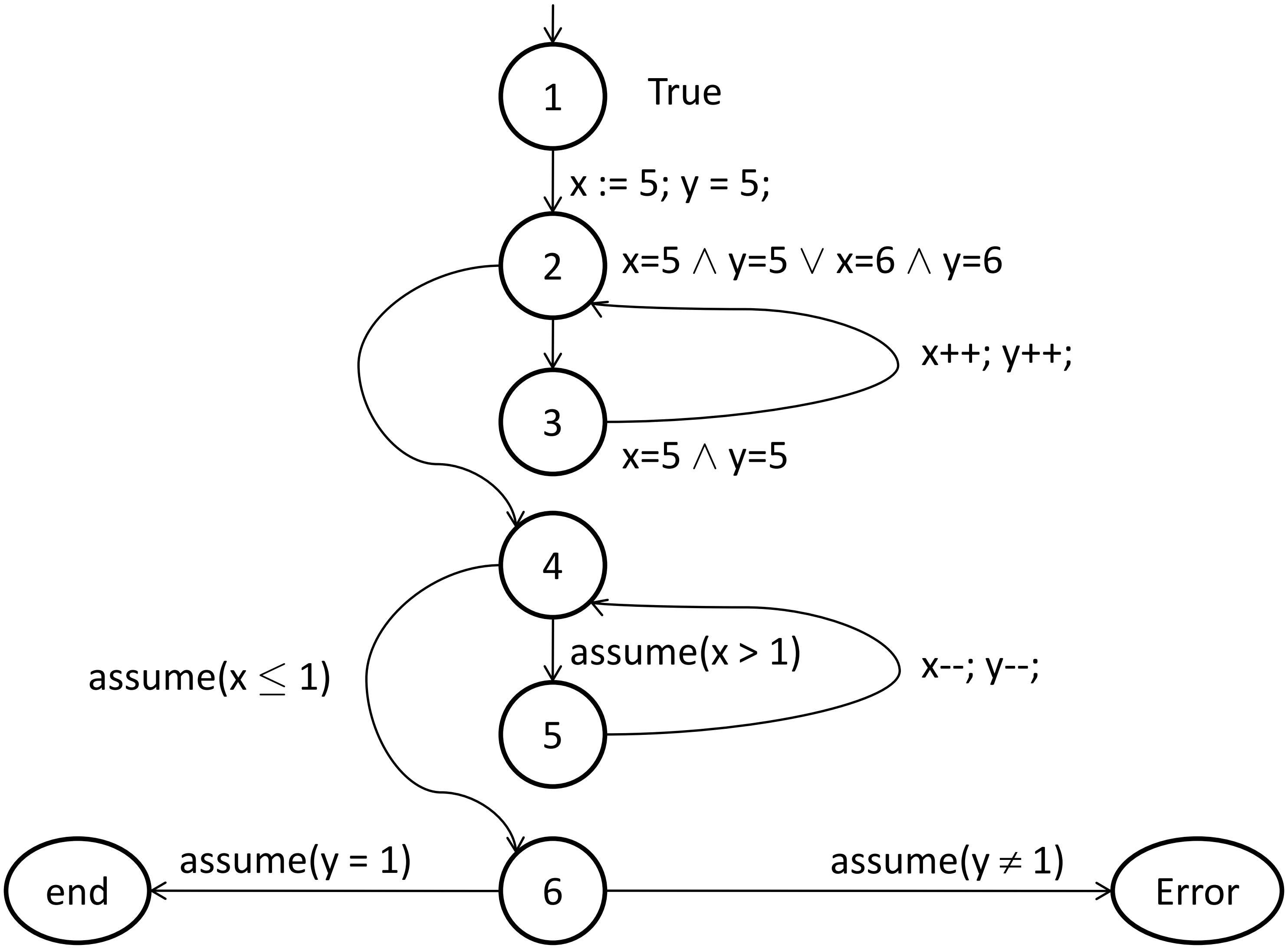


1
True
x := 5; y = 5;
2
x=5 ∧ y=5
x++; y++;
3
4
assume(x > 1)
x--; y--;
5
assume(x ≤ 1)
end
assume(y = 1)
6
assume(y ≠ 1)
Error

post#

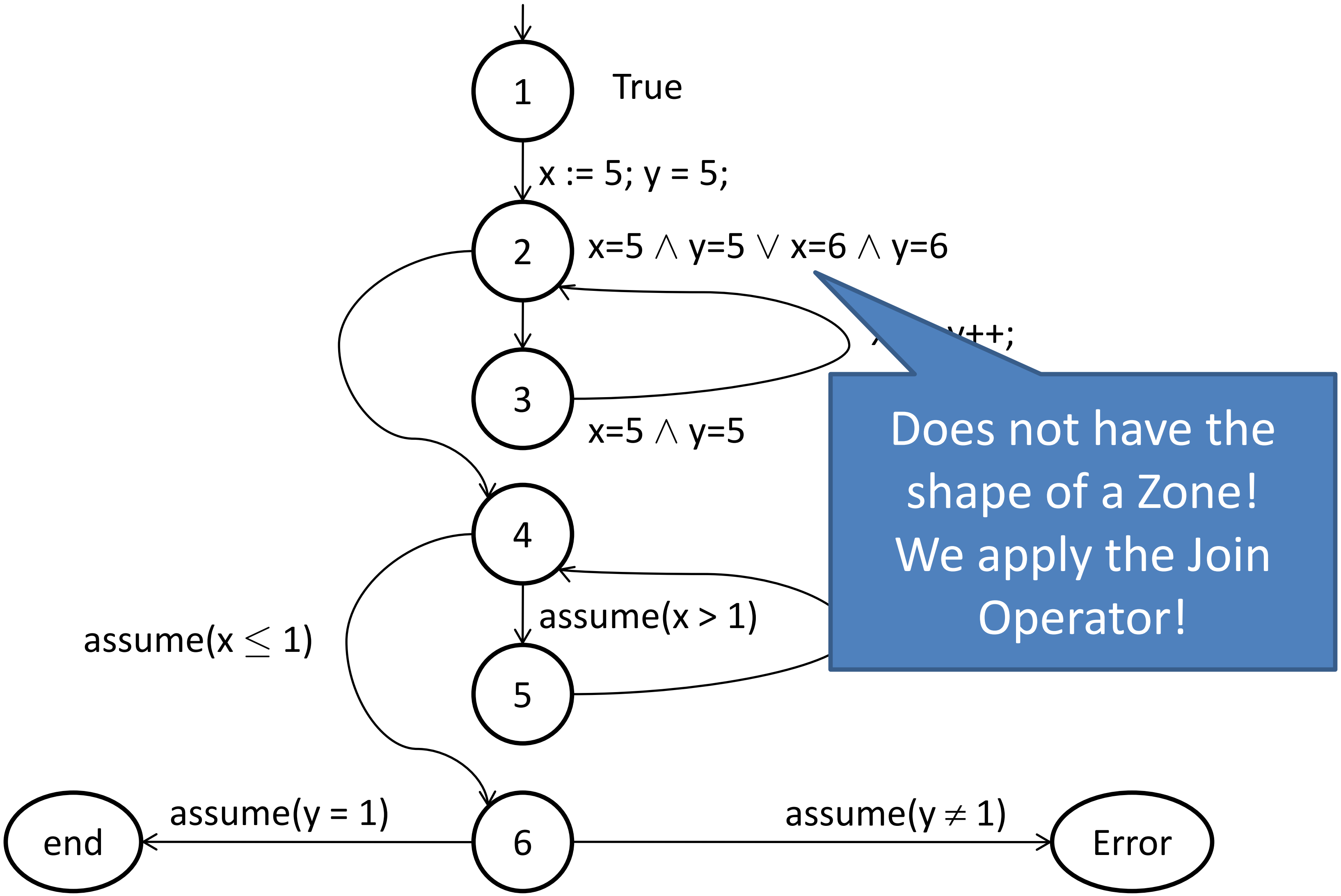

1
True
x := 5; y = 5;
2
x=5 ∧ y=5
x++; y++;
3
x=5 ∧ y=5
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$\mathbf{post}^{\#}$



1
True
x := 5; y = 5;
2
x=5 ∧ y=5 ∨ x=6 ∧ y=6
x++; y++;
3
x=5 ∧ y=5
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

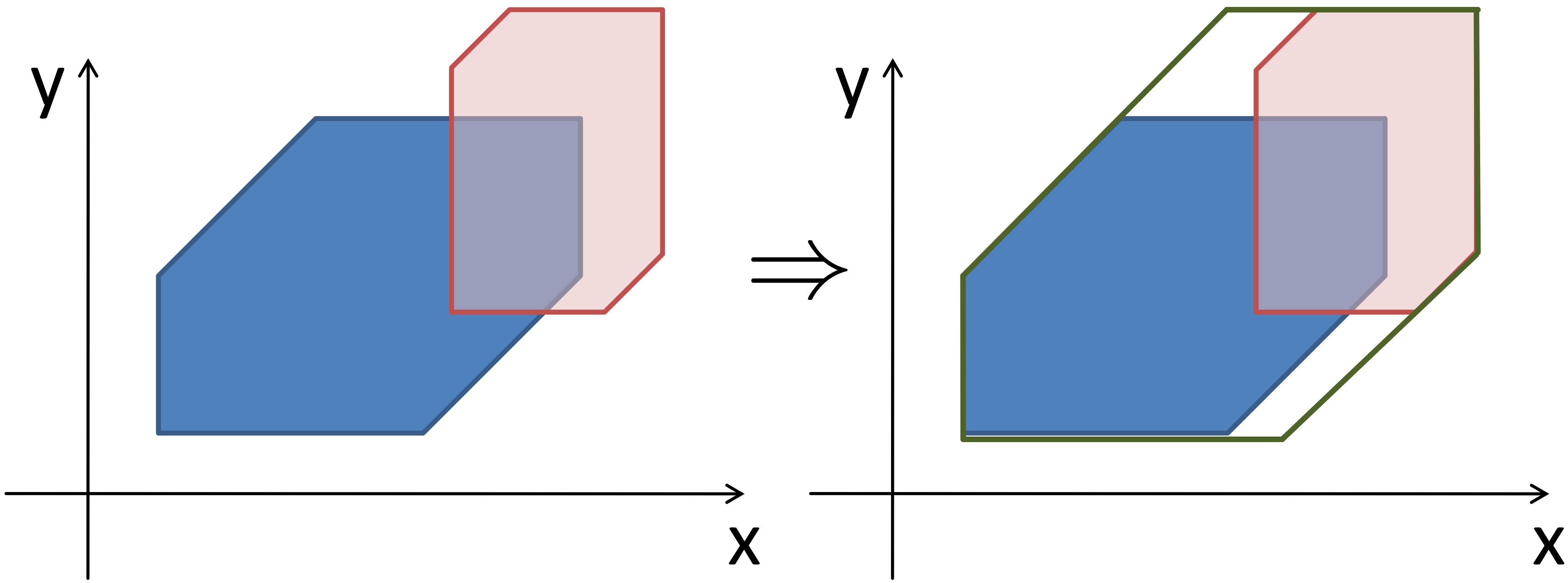
$\mathbf{post}^{\#}$



1
True
x := 5; y = 5;
2
x=5 ∧ y=5 ∨ x=6 ∧ y=6
3
x=5 ∧ y=5
4
assume(x > 1)
assume(x ≤ 1)
5
6
assume(y = 1)
end
assume(y ≠ 1)
Error
Does not have the shape of a Zone! We apply the Join Operator!

Join

$Zone_1 \sqcup Zone_2$ = $Zone$

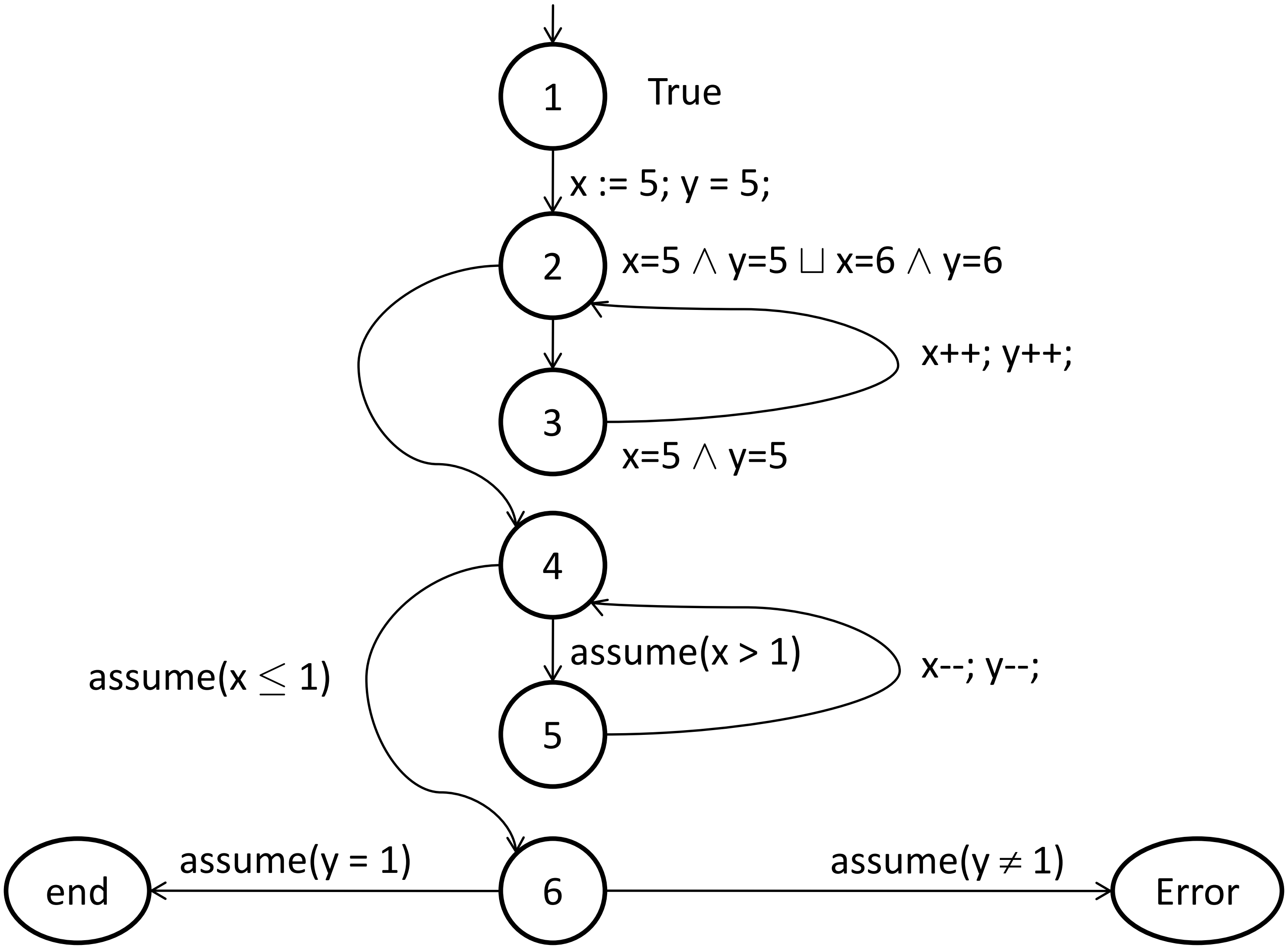


$x \leq c_1 \wedge -x \leq c_2 \wedge y \leq c_3 \wedge -y \leq c_4 \wedge$
$x-y \leq c_5 \wedge y-x \leq c_6 \sqcup$
$x \leq d_1 \wedge -x \leq d_2 \wedge y \leq d_3 \wedge -y \leq d_4 \wedge$
$x-y \leq d_5 \wedge y-x \leq d_6,$
$c_1,\dots,c_8, d_1,\dots,d_8 \in \mathbb{Z} \cup \{\infty\}$

=

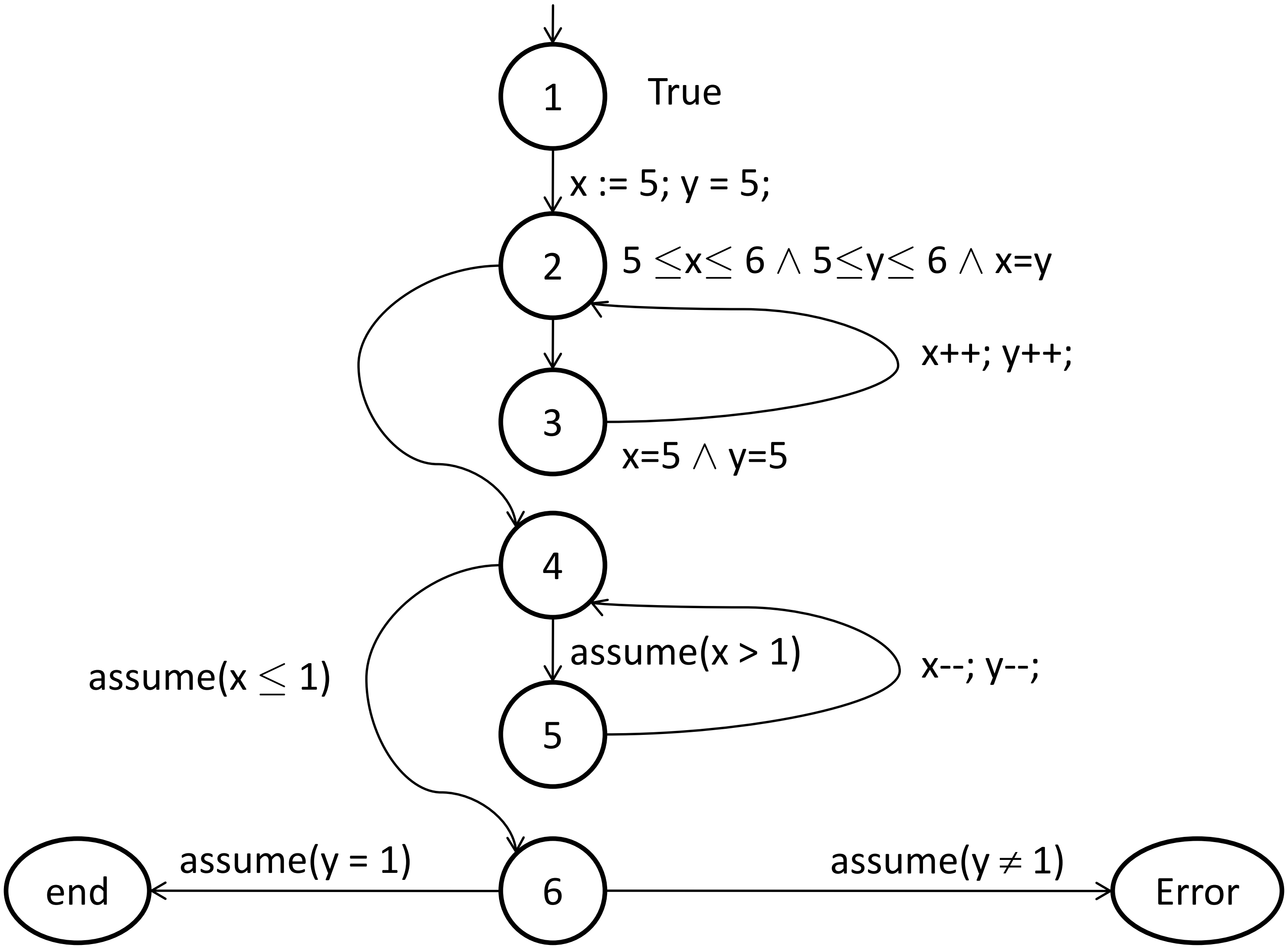
$x \leq \max\{c_1, d_1\} \wedge -x \leq \max\{c_2, d_2\} \wedge$
$y \leq \max\{c_3, d_3\} \wedge -y \leq \max\{c_4, d_4\} \wedge$
$x-y \leq \max\{c_6, d_6\} \wedge y-x \leq \max\{c_7, d_7\}$

Join



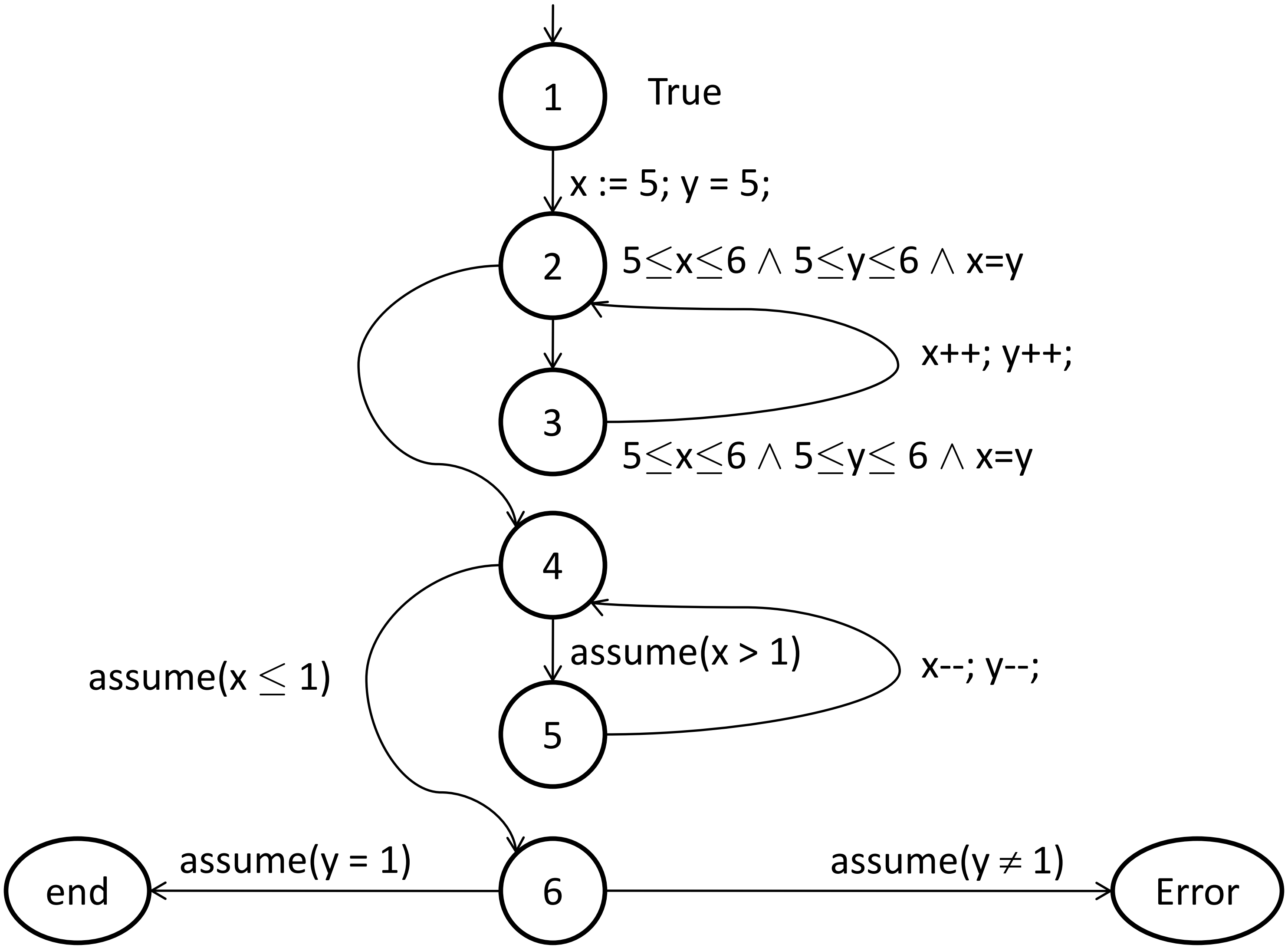
1
True
x := 5; y = 5;
2
x=5 ∧ y=5 ⊔ x=6 ∧ y=6
x++; y++;
3
x=5 ∧ y=5
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
6
end
assume(y = 1)
assume(y ≠ 1)
Error

Join



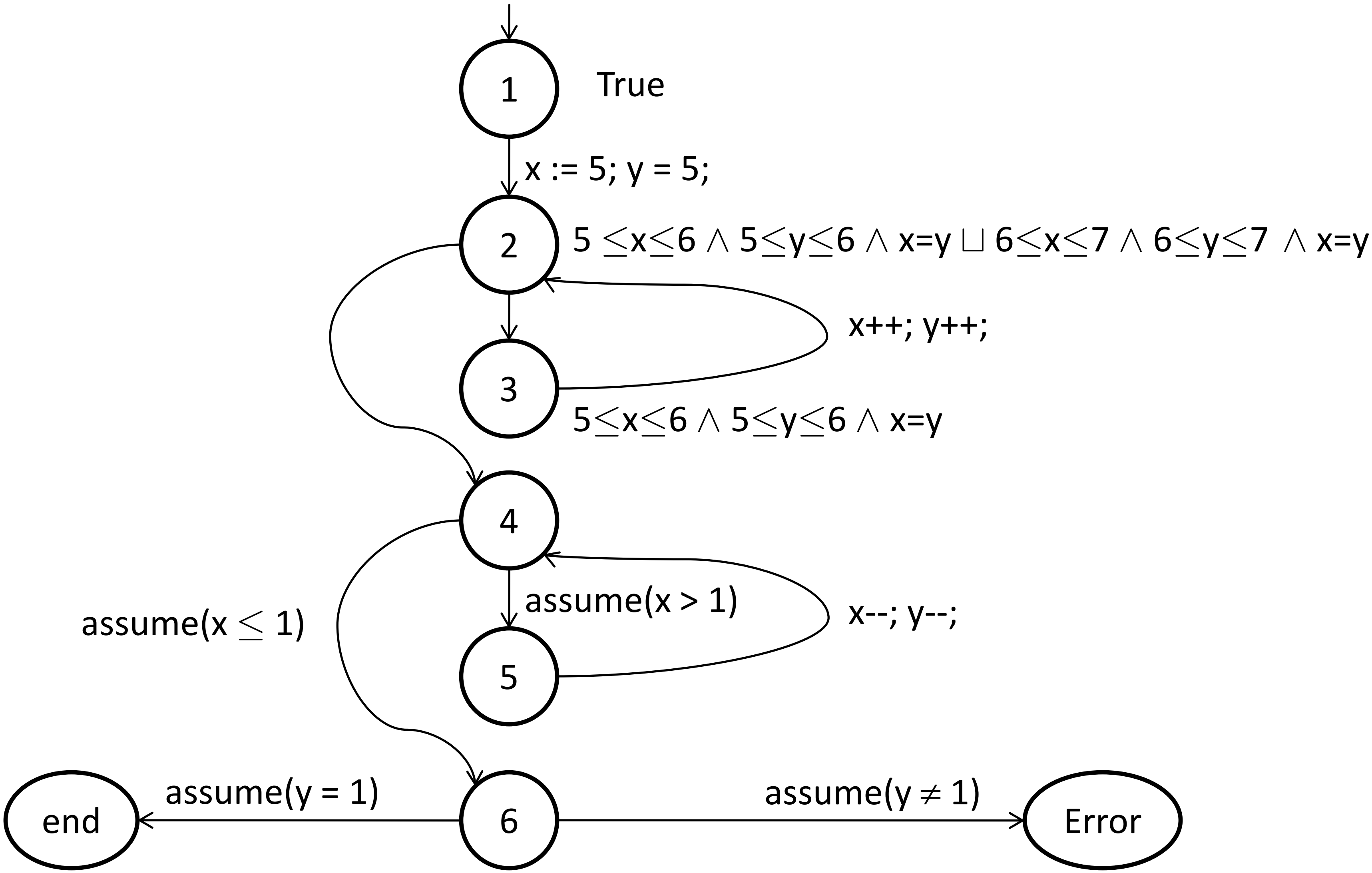
1
True
x := 5; y = 5;
2
5 ≤x≤ 6 ∧ 5≤y≤ 6 ∧ x=y
x++; y++;
3
x=5 ∧ y=5
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$post^{\#}$



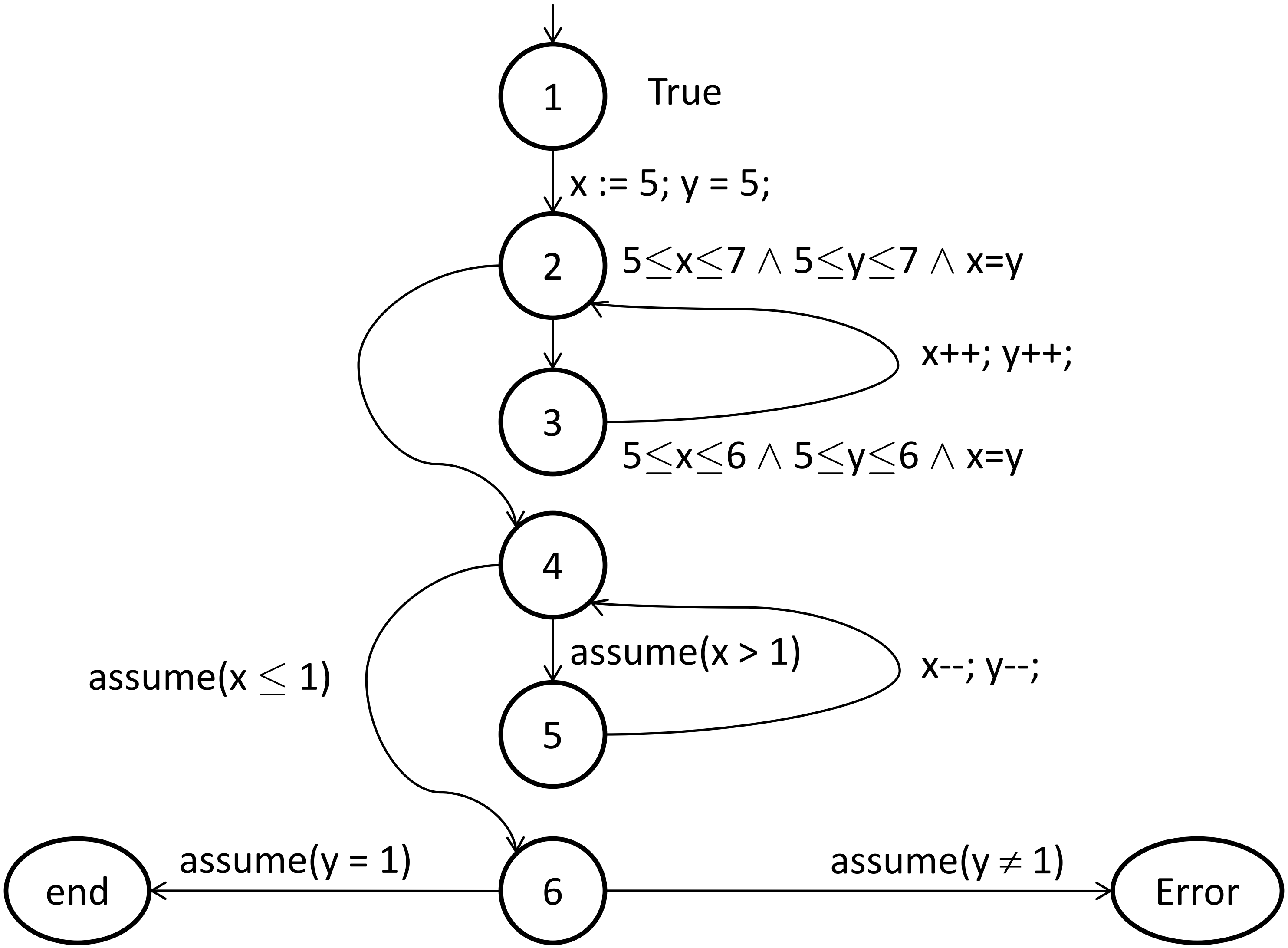
1
True
x := 5; y = 5;
2
5≤x≤6 ∧ 5≤y≤6 ∧ x=y
x++; y++;
3
5≤x≤6 ∧ 5≤y≤ 6 ∧ x=y
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
assume(y = 1)
end
6
assume(y ≠ 1)
Error

$\mathbf{post}^\#$

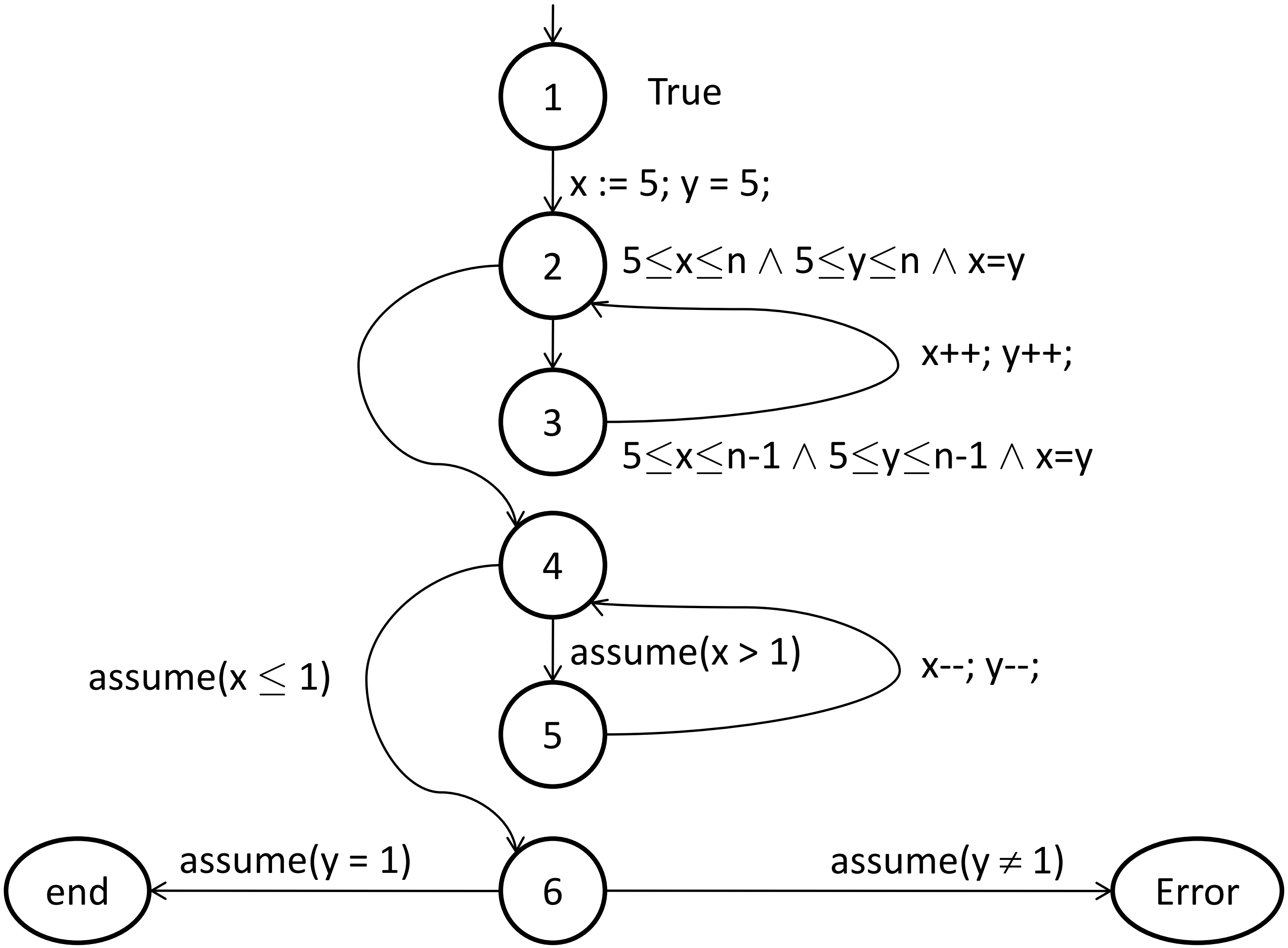


1
True
x := 5; y = 5;
2
5 ≤x≤6 ∧ 5≤y≤6 ∧ x=y ⊔ 6≤x≤7 ∧ 6≤y≤7 ∧ x=y
x++; y++;
3
5≤x≤6 ∧ 5≤y≤6 ∧ x=y
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
assume(y = 1)
6
assume(y ≠ 1)
end
Error

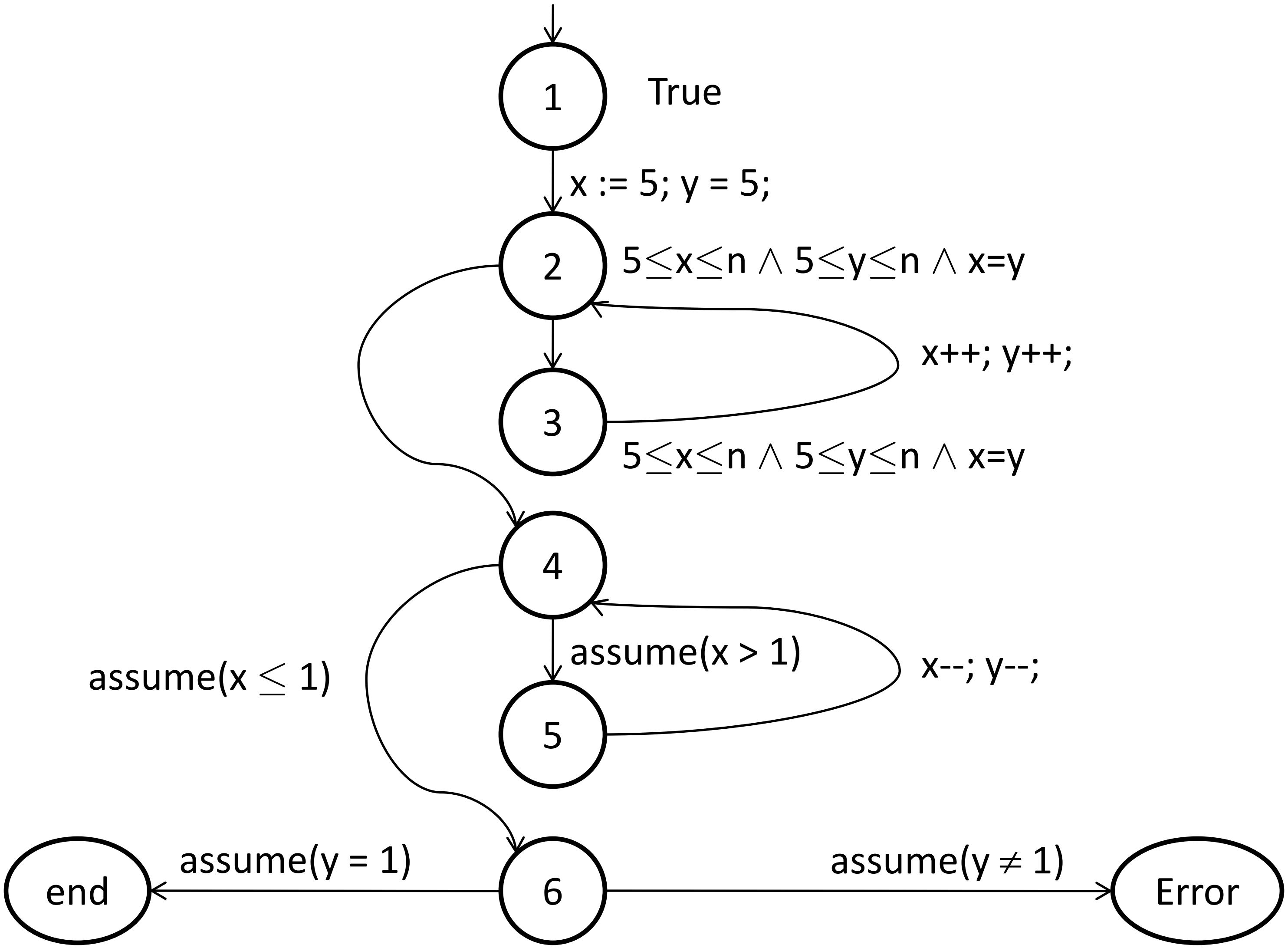
Join leads to Divergence



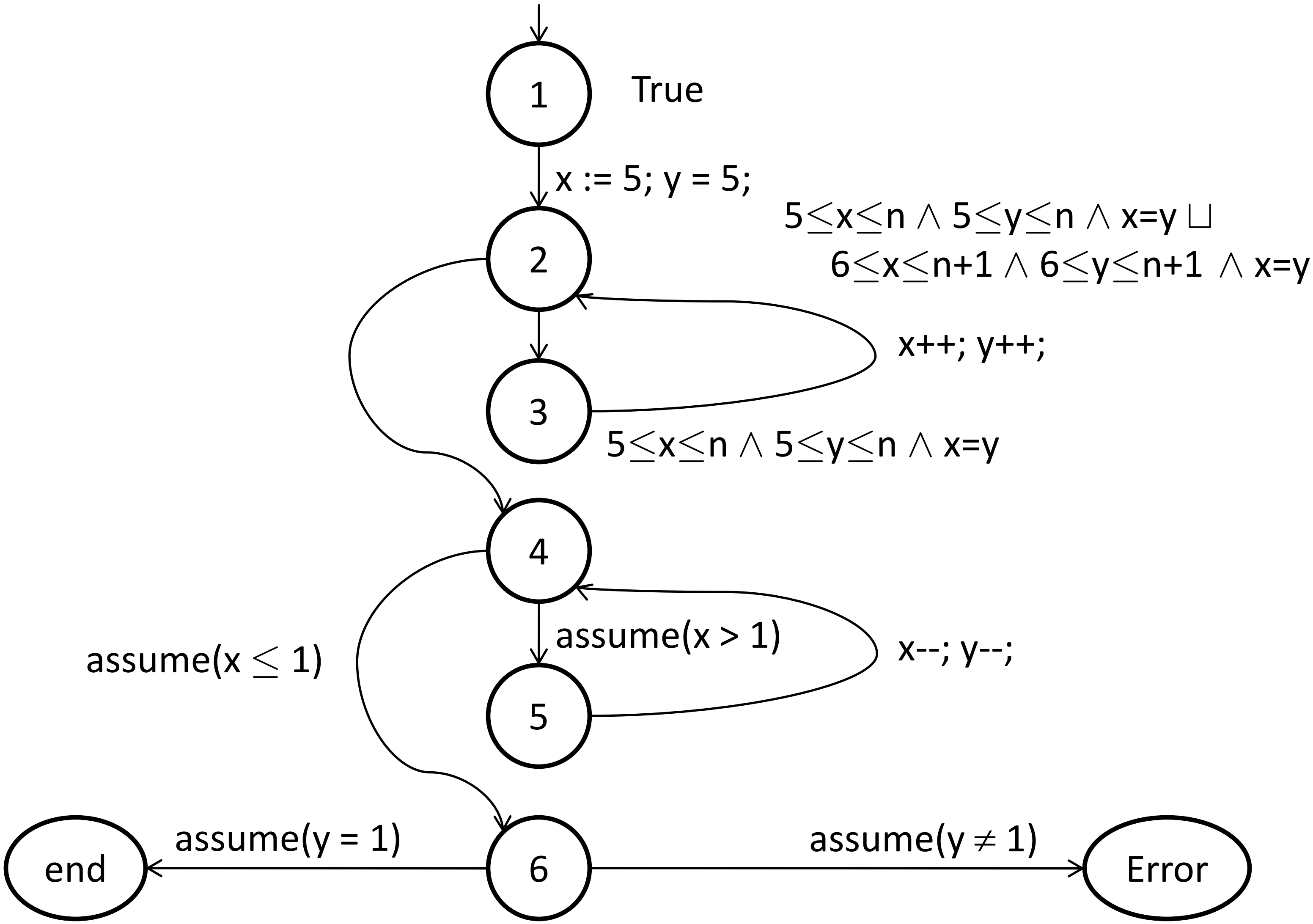
Join leads to Divergence



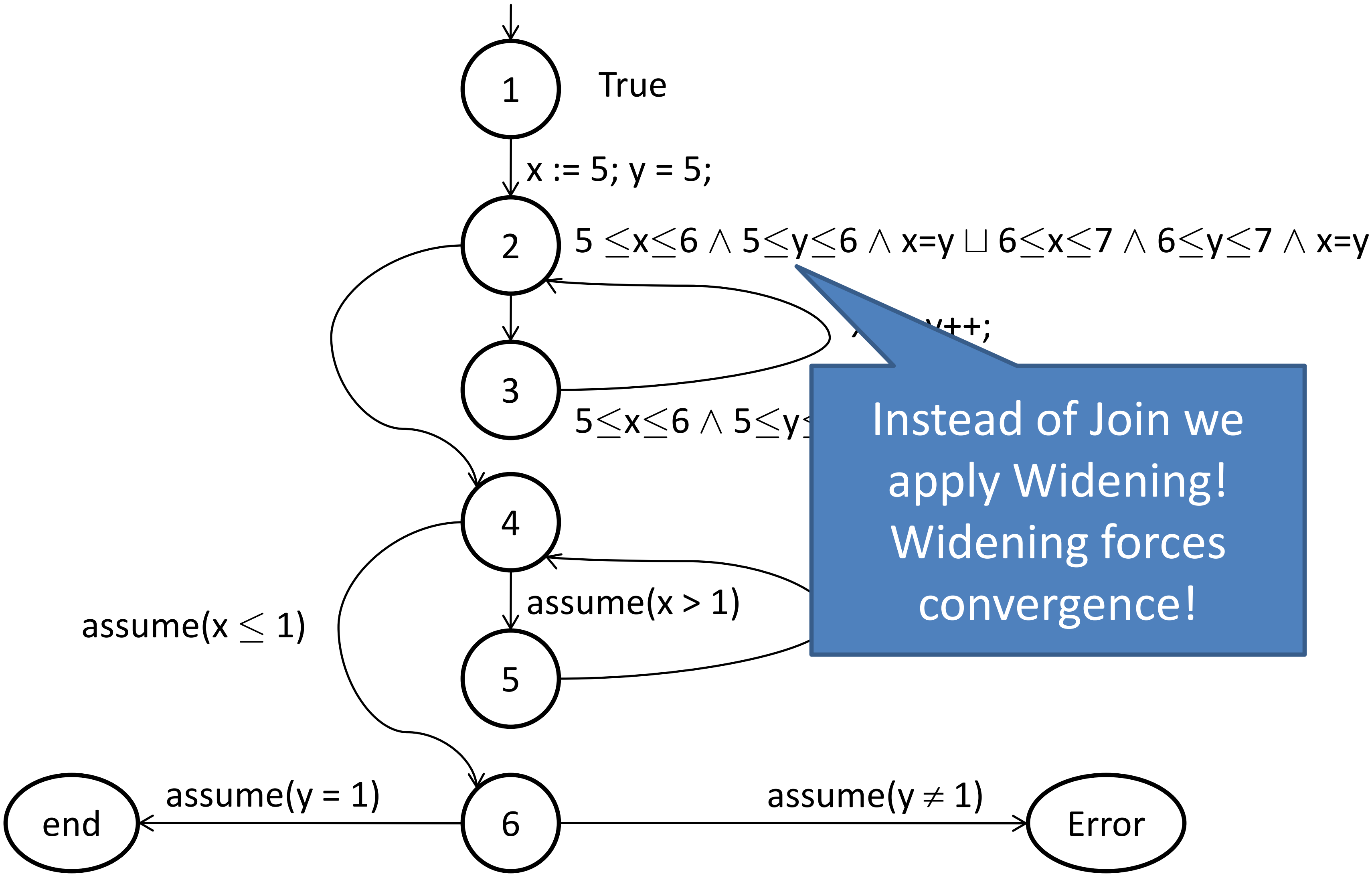
Join leads to Divergence



Join leads to Divergence



Widening



1
True
x := 5; y = 5;
2
5 ≤x≤6 ∧ 5≤y≤6 ∧ x=y ⊔ 6≤x≤7 ∧ 6≤y≤7 ∧ x=y
y++;
3
5≤x≤6 ∧ 5≤y
4
assume(x > 1)
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error
Instead of Join we apply Widening! Widening forces convergence!

Widening

Idea:

Drop the inequalities which are not stable after one iteration!

$5 \leq x \leq 6 \wedge 5 \leq y \leq 6 \wedge x=y \sqcup 6 \leq x \leq 7 \wedge 6 \leq y \leq 7 \wedge x=y$

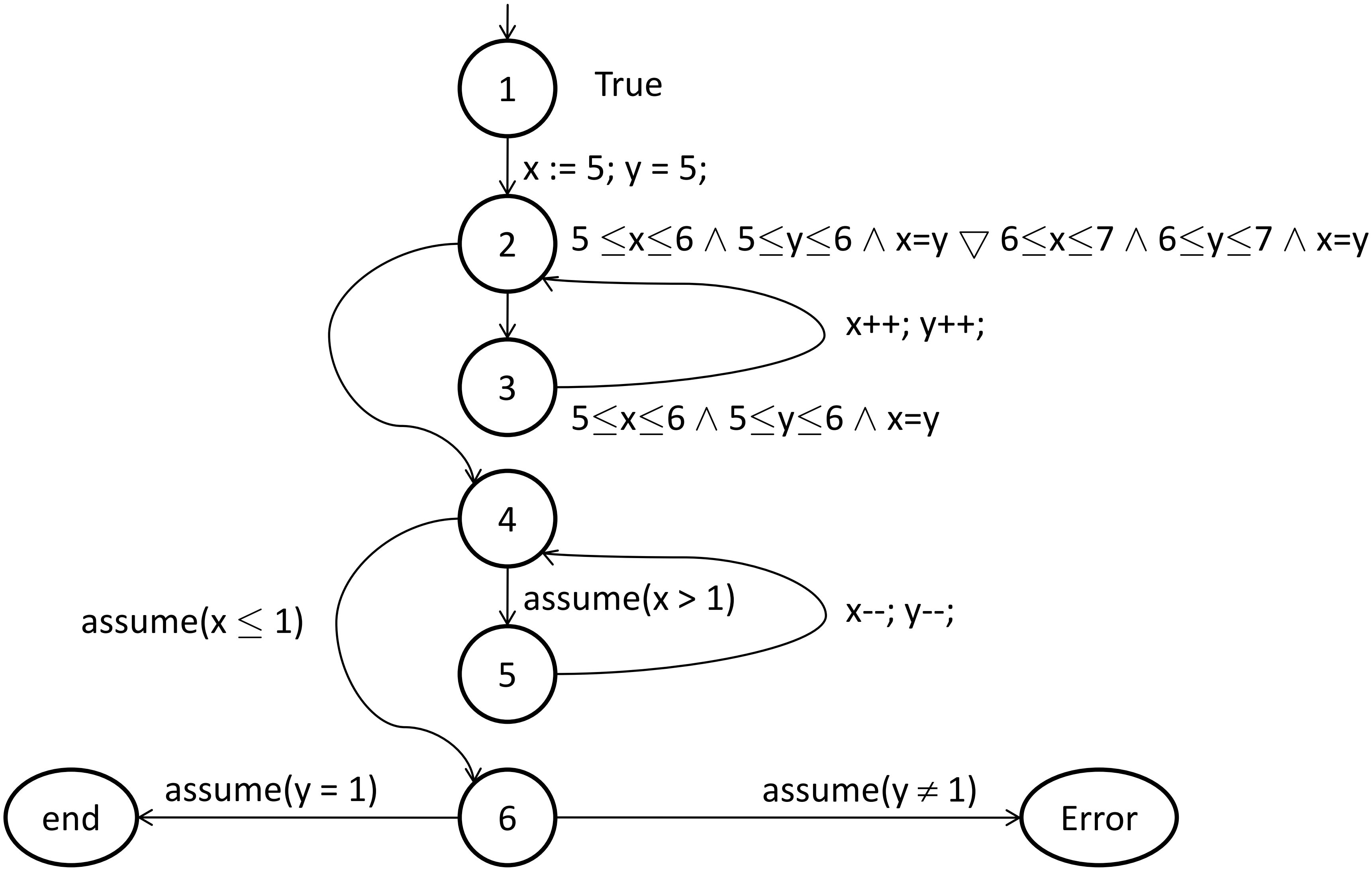
$= 5 \leq x \leq 7 \wedge 5 \leq y \leq 7 \wedge x=y$

We see, that $x \leq 6$ and $y \leq 6$ are not stable, therefore we drop them!

$5 \leq x \leq 6 \wedge 5 \leq y \leq 6 \wedge x=y \triangledown 6 \leq x \leq 7 \wedge 6 \leq y \leq 7 \wedge x=y =$

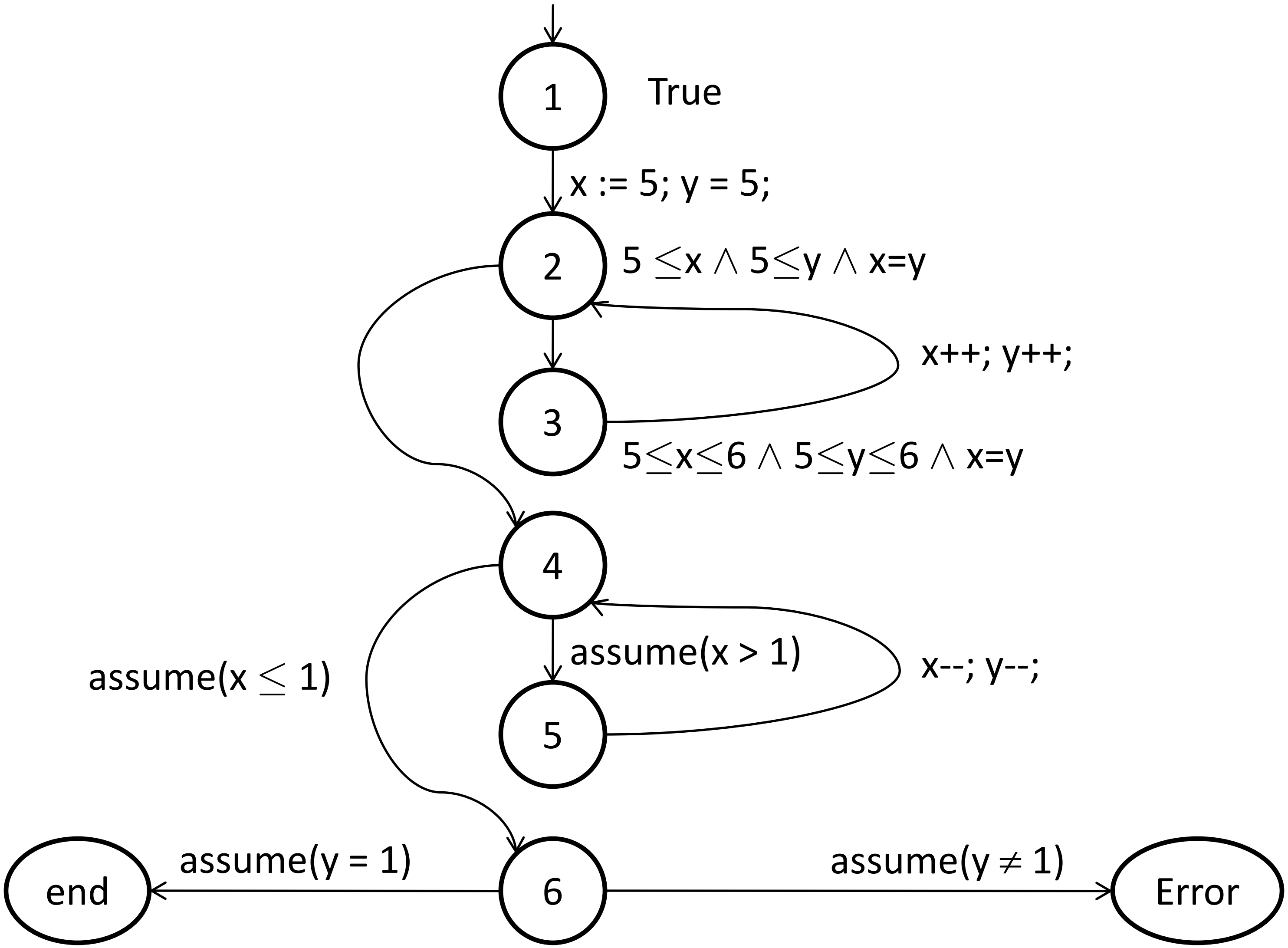
$5 \leq x \wedge 5 \leq y \wedge x=y$

Widening



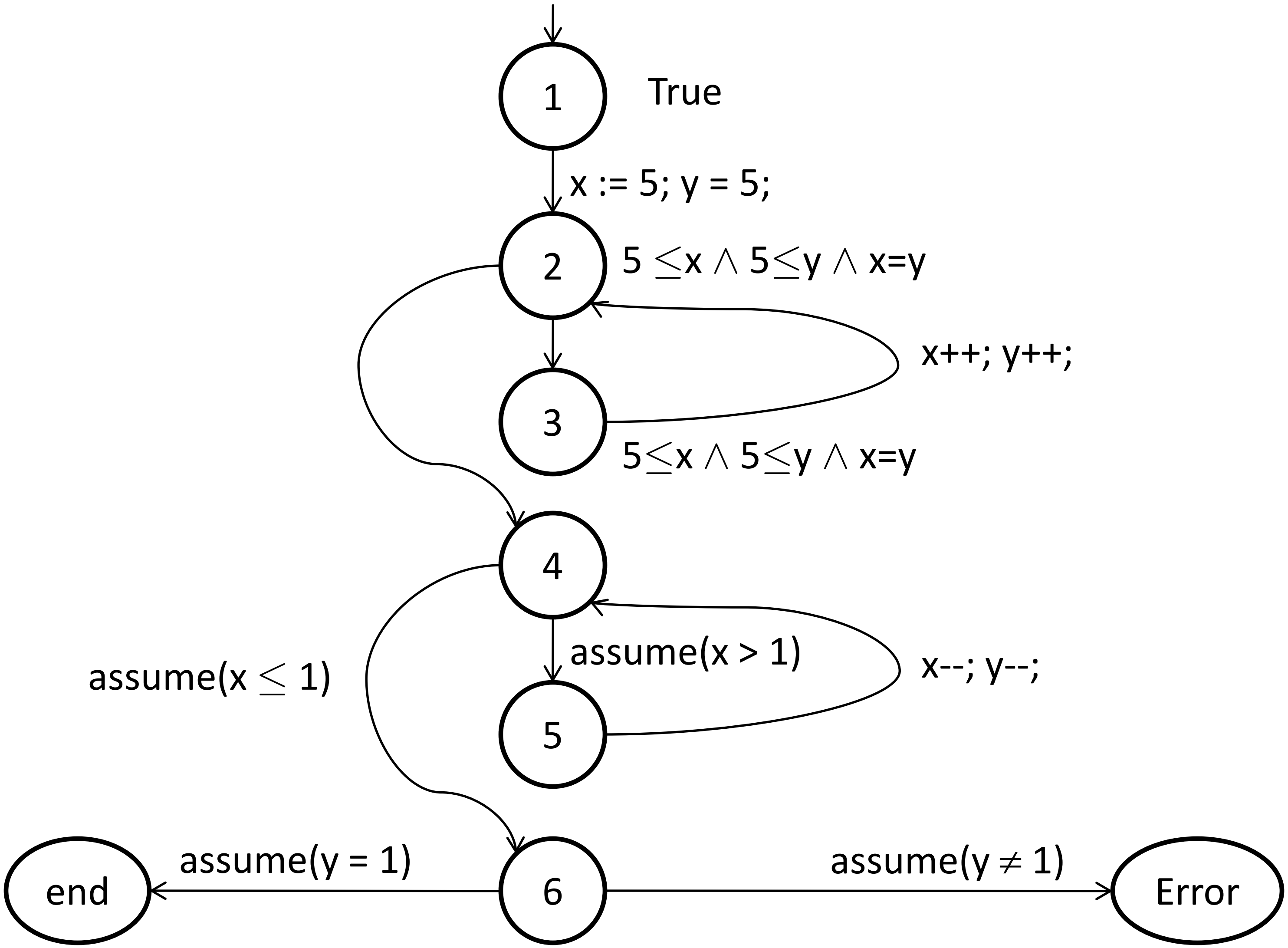
1
True
x := 5; y = 5;
2
5 ≤x≤6 ∧ 5≤y≤6 ∧ x=y ▽ 6≤x≤7 ∧ 6≤y≤7 ∧ x=y
x++; y++;
3
5≤x≤6 ∧ 5≤y≤6 ∧ x=y
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Widening



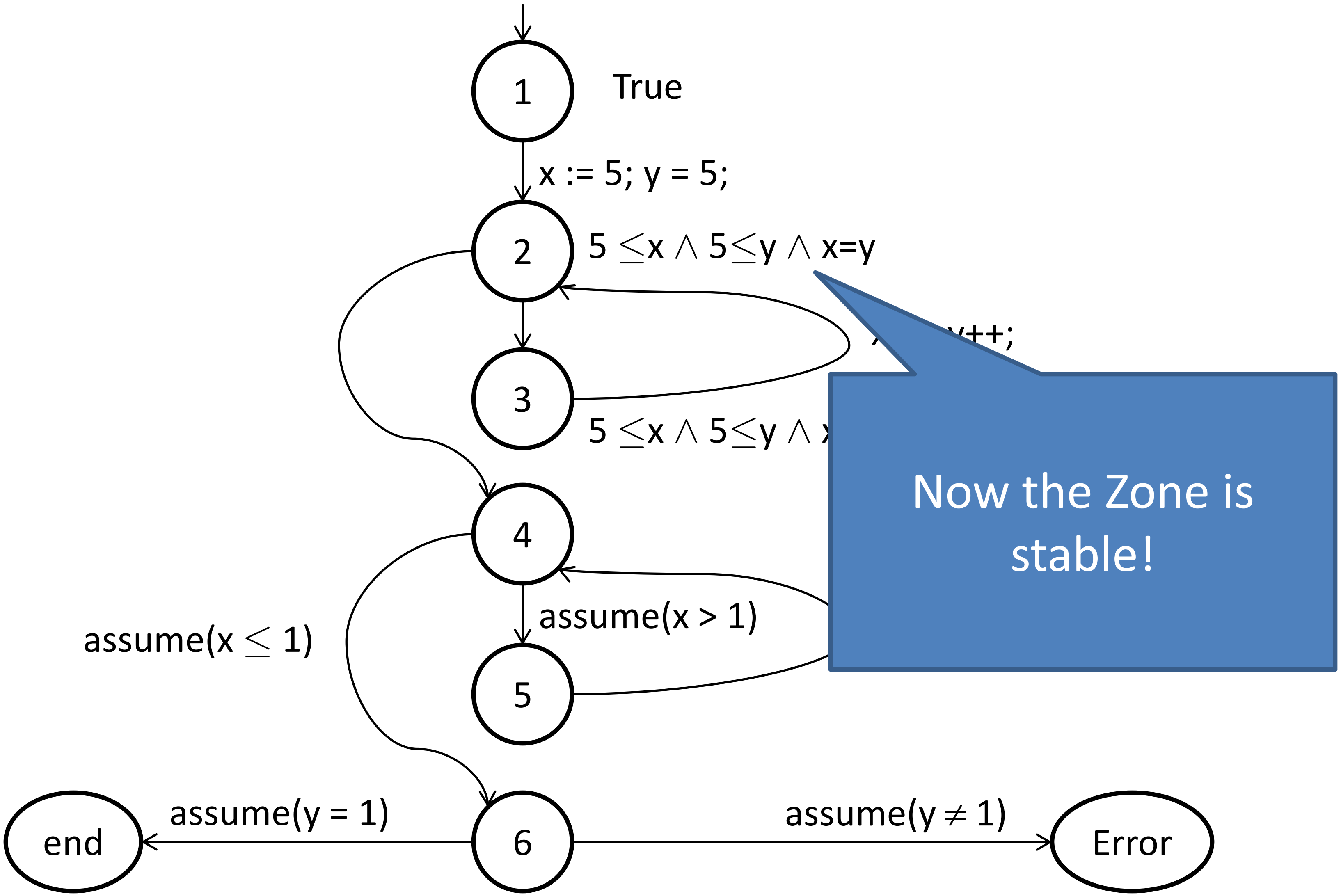
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5≤x≤6 ∧ 5≤y≤6 ∧ x=y
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$post^{\#}$

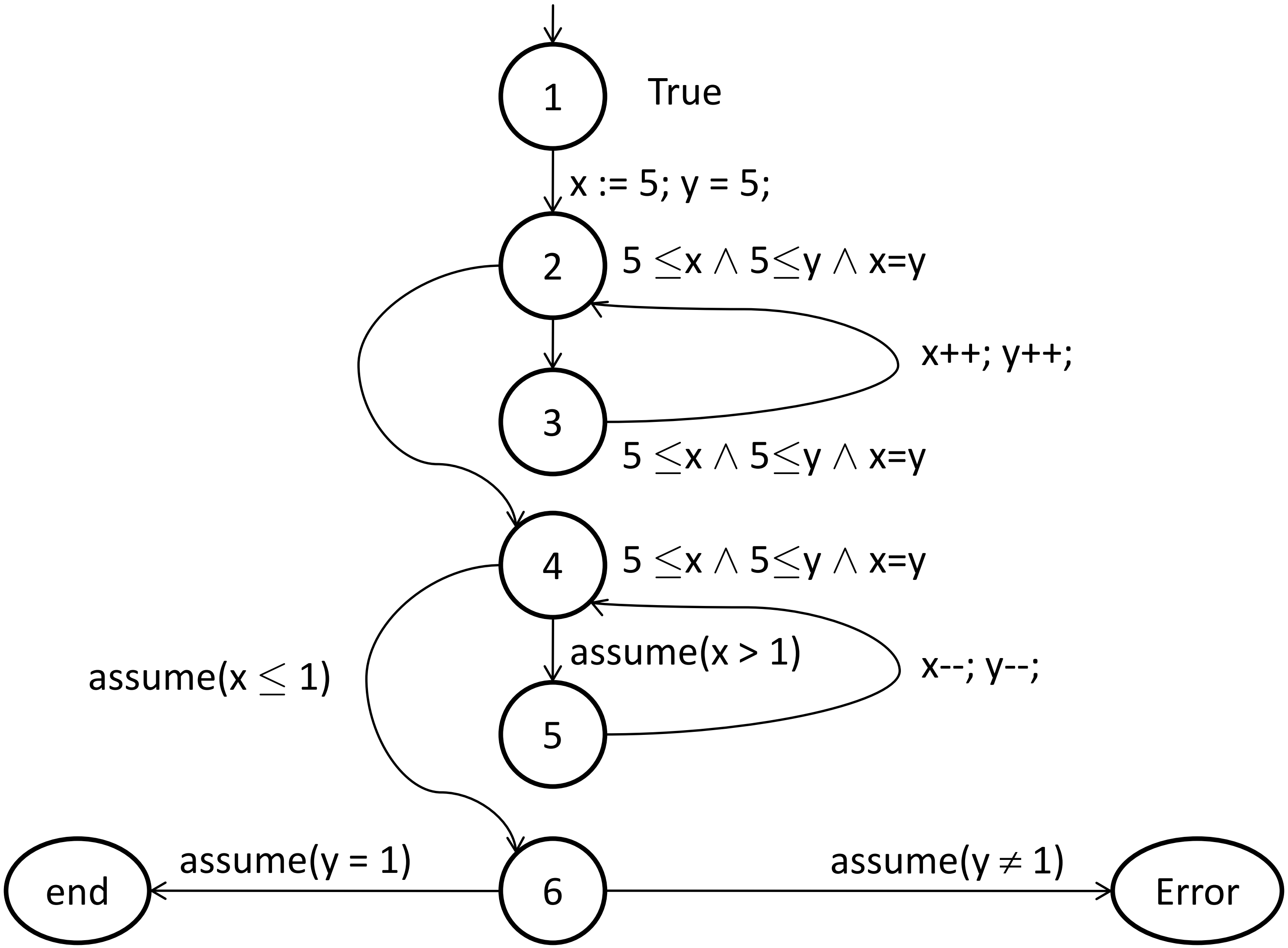


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5≤x ∧ 5≤y ∧ x=y
4
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

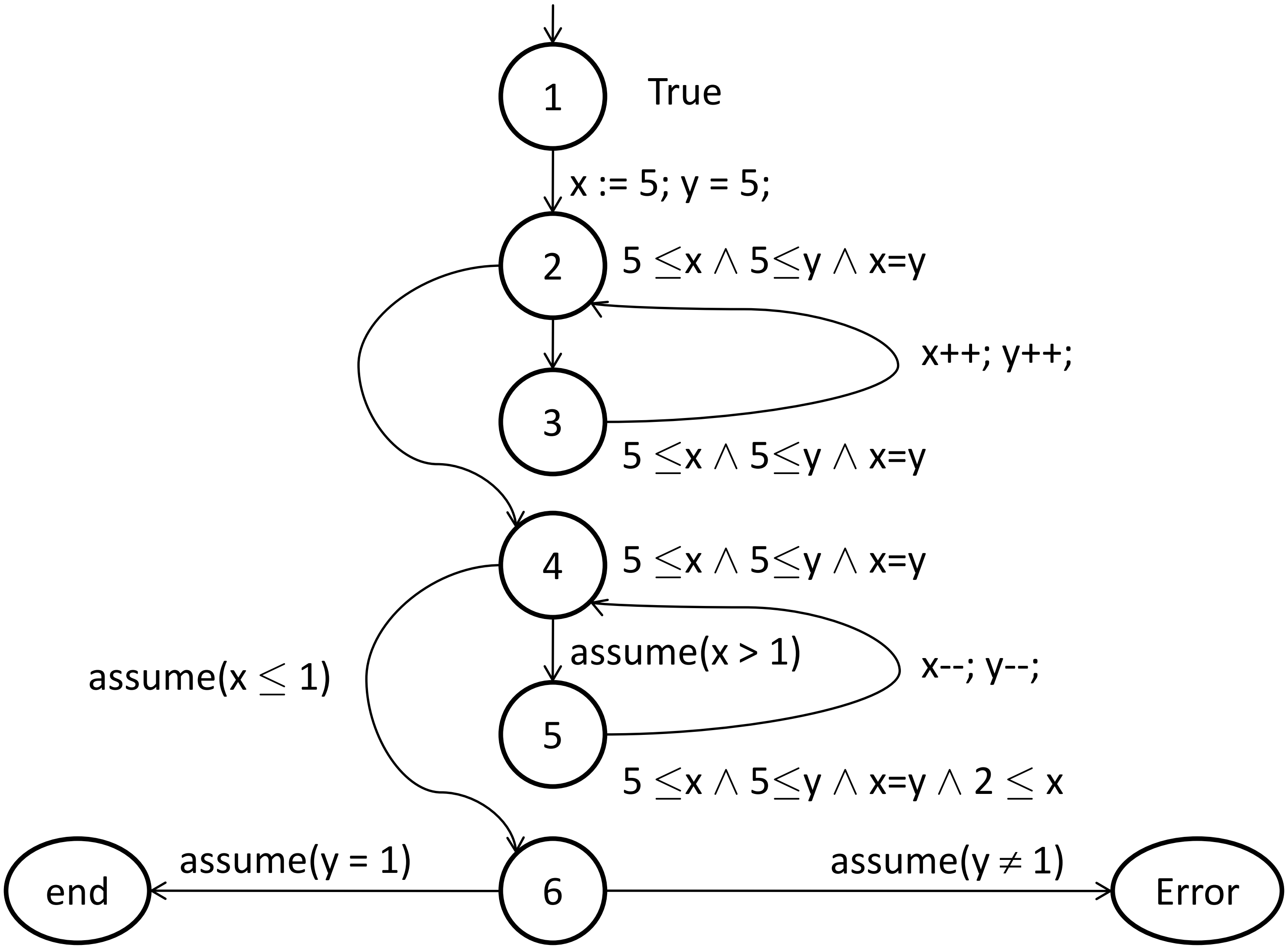
Zone is stable



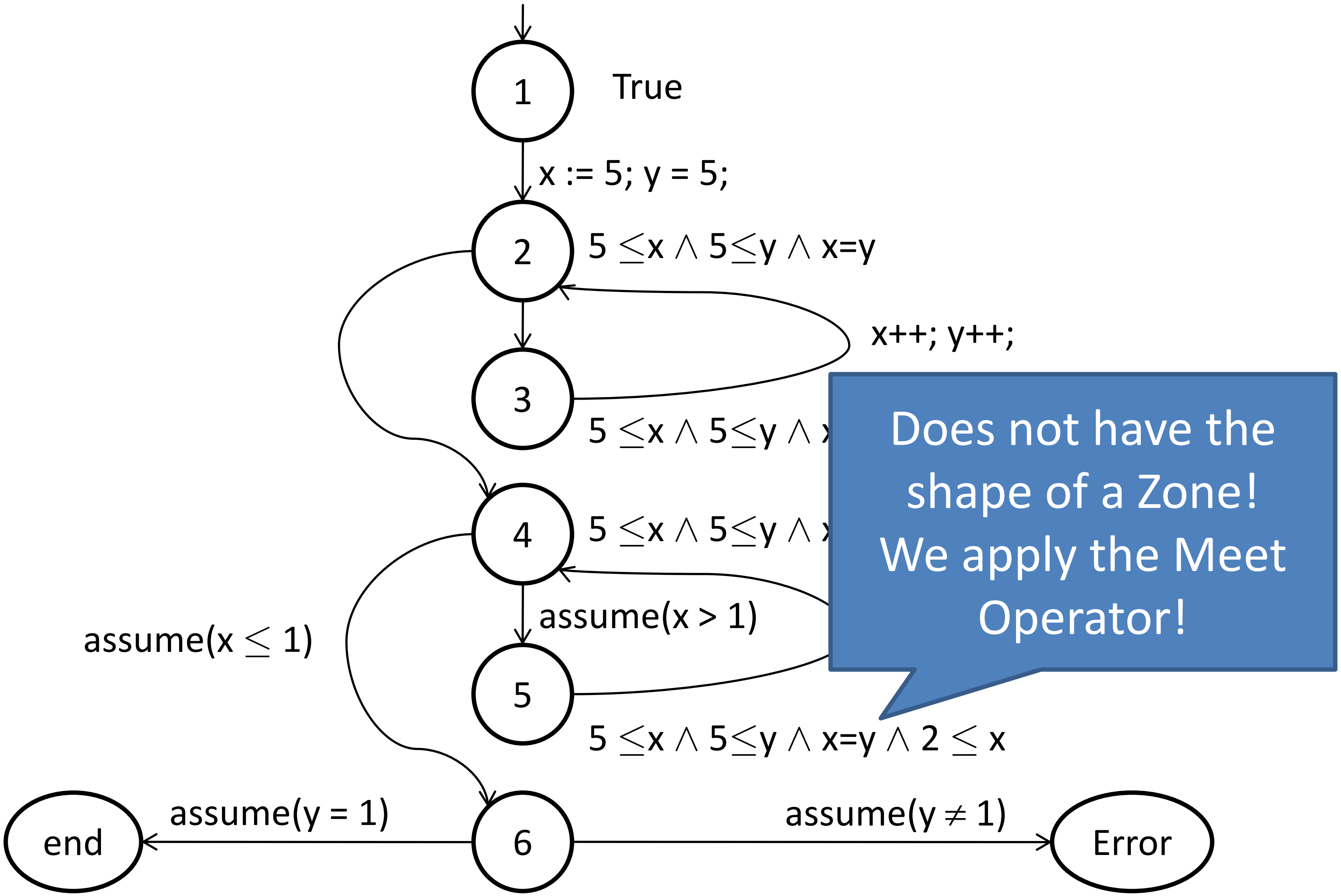
$\text{post}^{\#}$


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
5 ≤x ∧ 5≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Meet

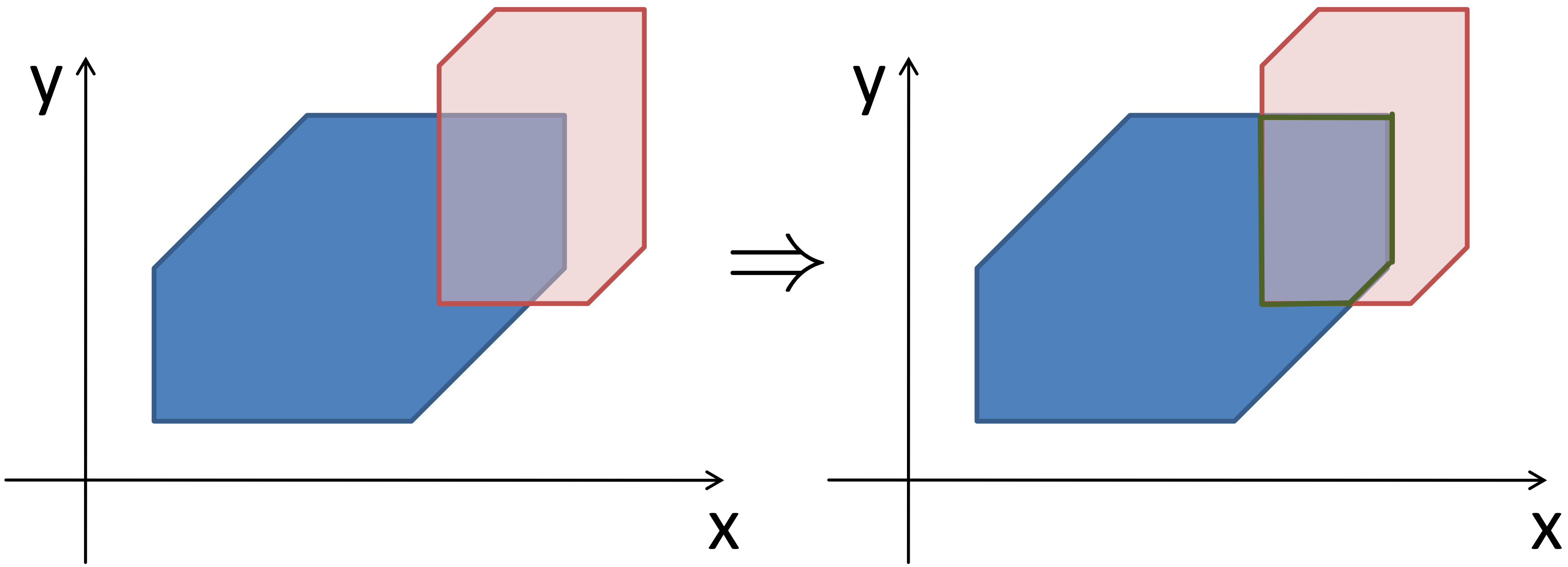

1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
5 ≤x ∧ 5≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y ∧ 2 ≤ x
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Meet


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x
4
5 ≤x ∧ 5≤y ∧ x
assume(x > 1)
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y ∧ 2 ≤ x
end
assume(y = 1)
6
assume(y ≠ 1)
Error
Does not have the shape of a Zone!
We apply the Meet Operator!

Meet

Zone $\sqcap$ Zone = Zone

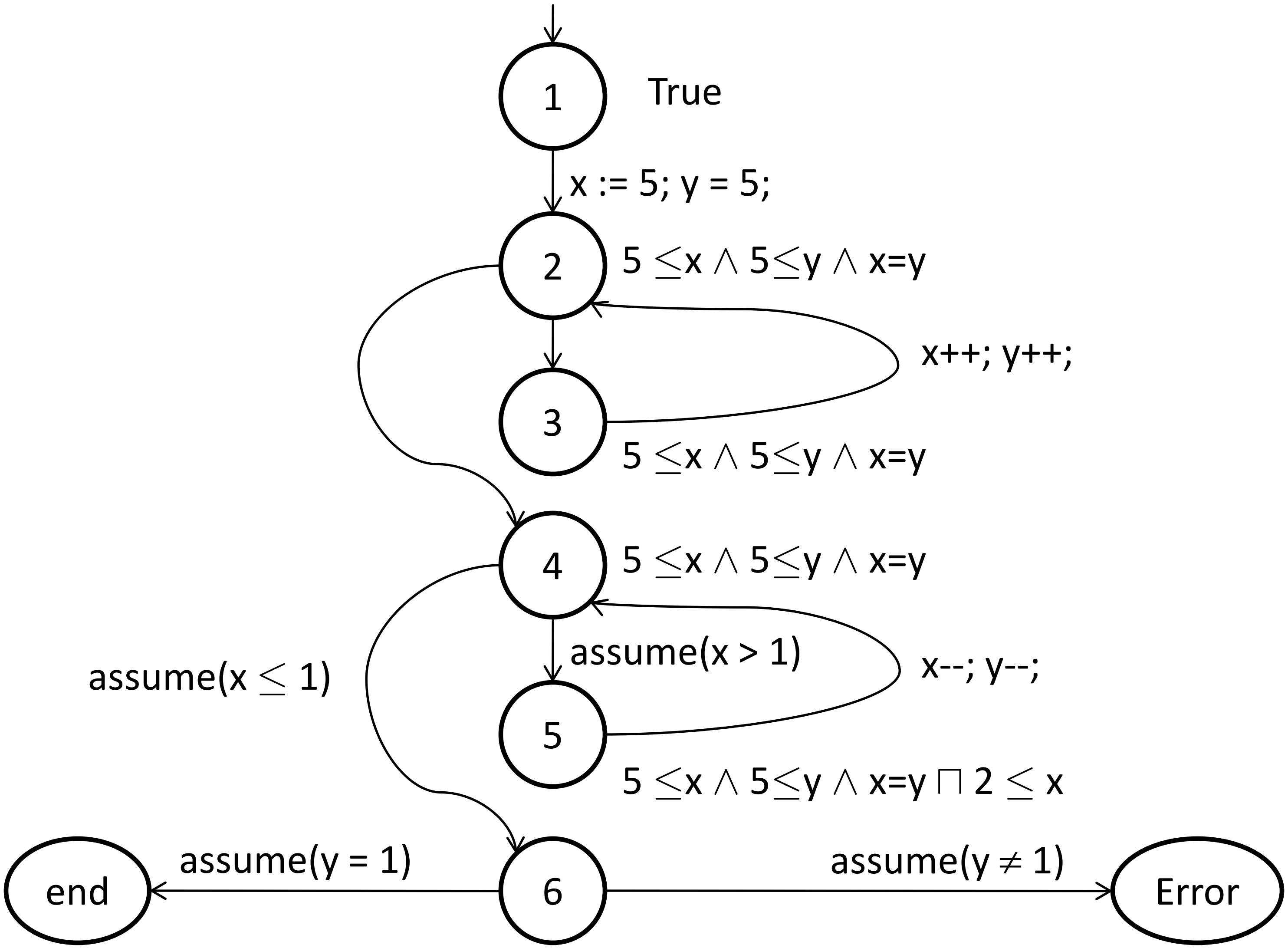


$x \leq c_1 \wedge -x \leq c_2 \wedge y \leq c_3 \wedge -y \leq c_4 \wedge$
$x-y \leq c_5 \wedge y-x \leq c_6 \sqcap$
$x \leq d_1 \wedge -x \leq d_2 \wedge y \leq d_3 \wedge -y \leq d_4 \wedge$
$x-y \leq d_5 \wedge y-x \leq d_6,$
$c_1,\ldots,c_6,d_1,\ldots,d_6 \in \mathbb{Z} \cup \{\infty\}$

=

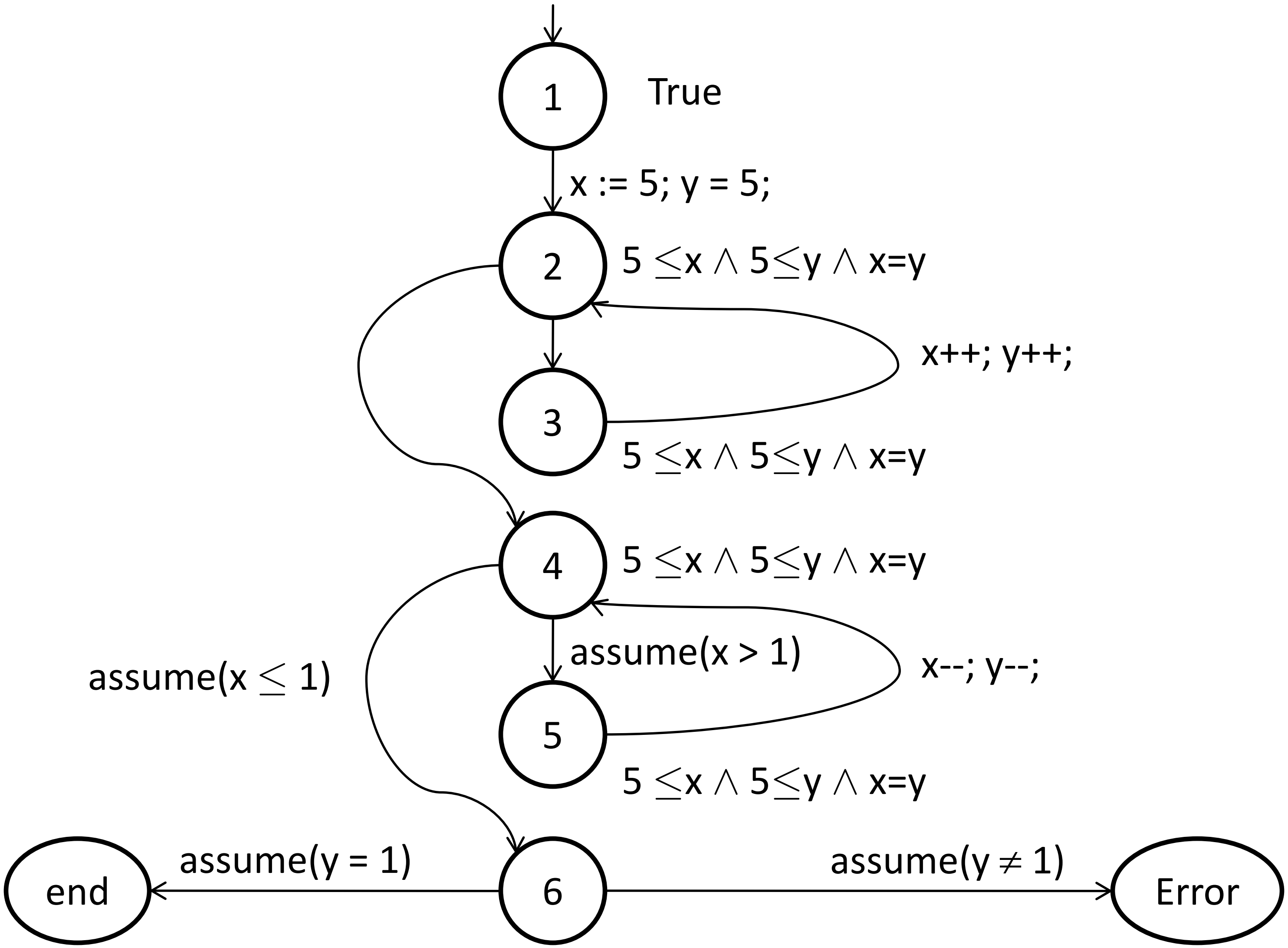
$x \leq \min\{c_1, d_1\} \wedge -x \leq \min\{c_2, d_2\} \wedge$
$y \leq \min\{c_3, d_3\} \wedge -y \leq \min\{c_4, d_4\} \wedge$
$x-y \leq \min\{c_5, d_5\} \wedge y-x \leq \min\{c_6, d_6\}$

Meet



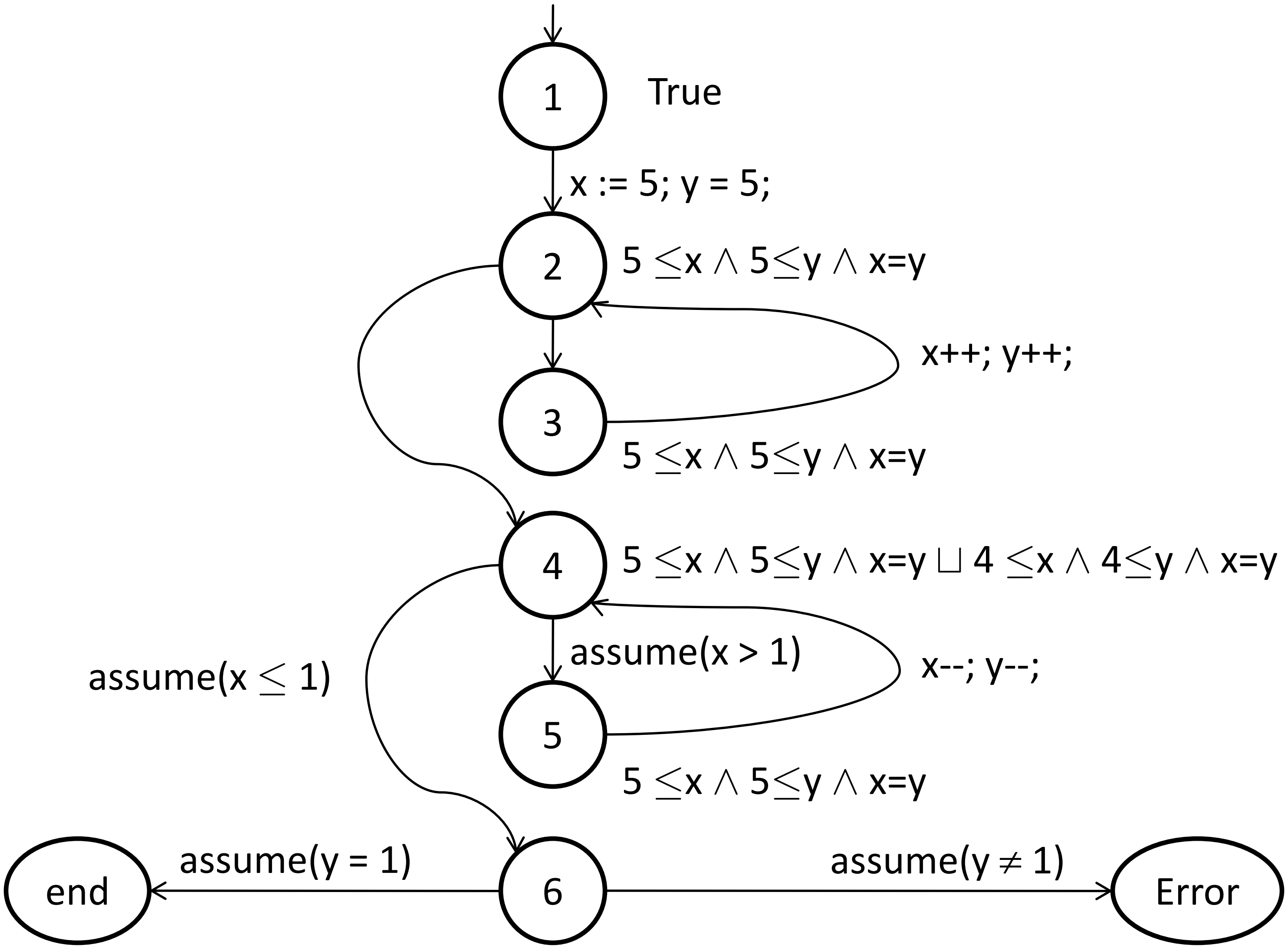
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
5 ≤x ∧ 5≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y ⊓ 2 ≤ x
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Meet



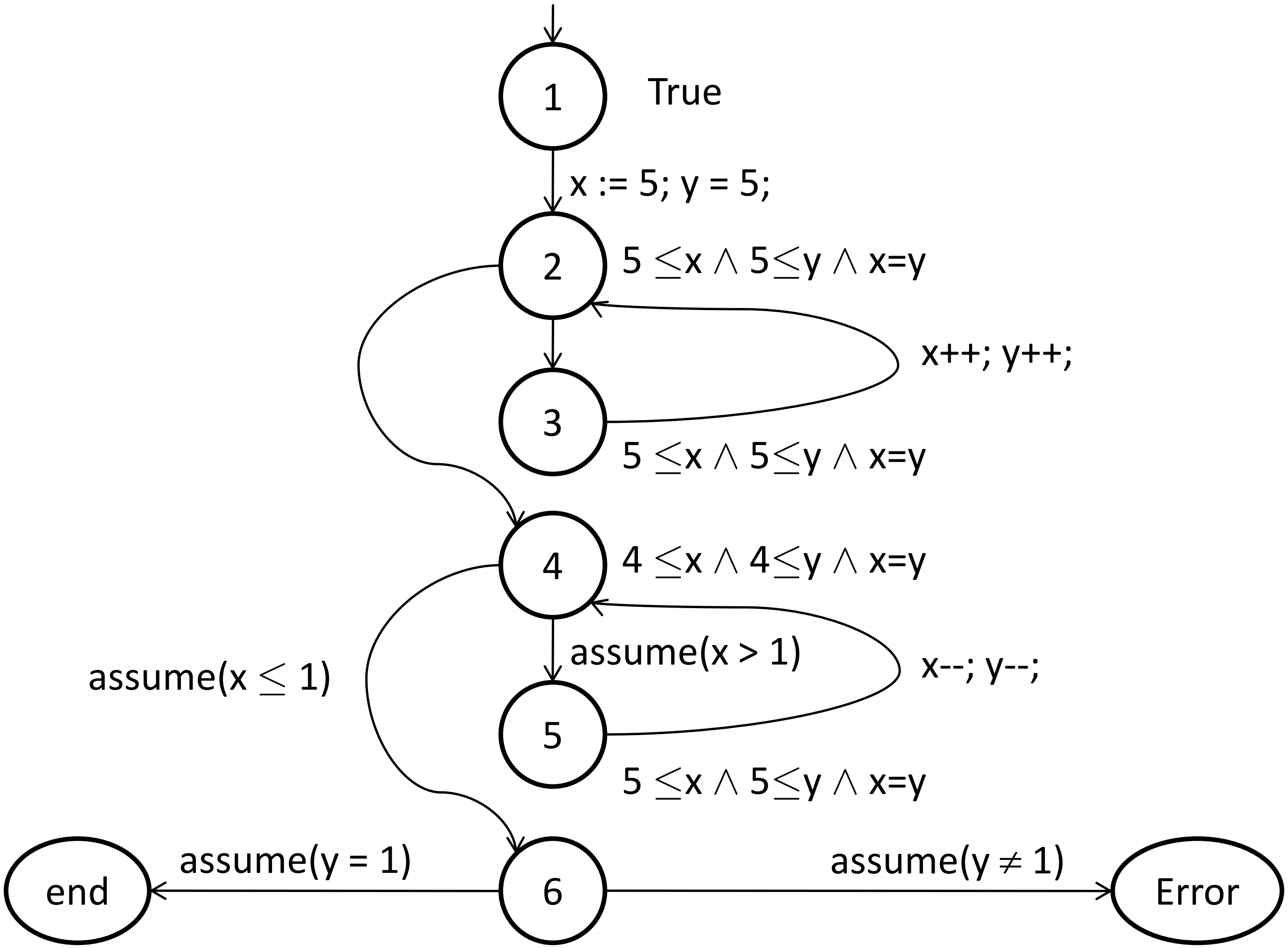
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
5 ≤x ∧ 5≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$\mathbf{post}^{\#}$



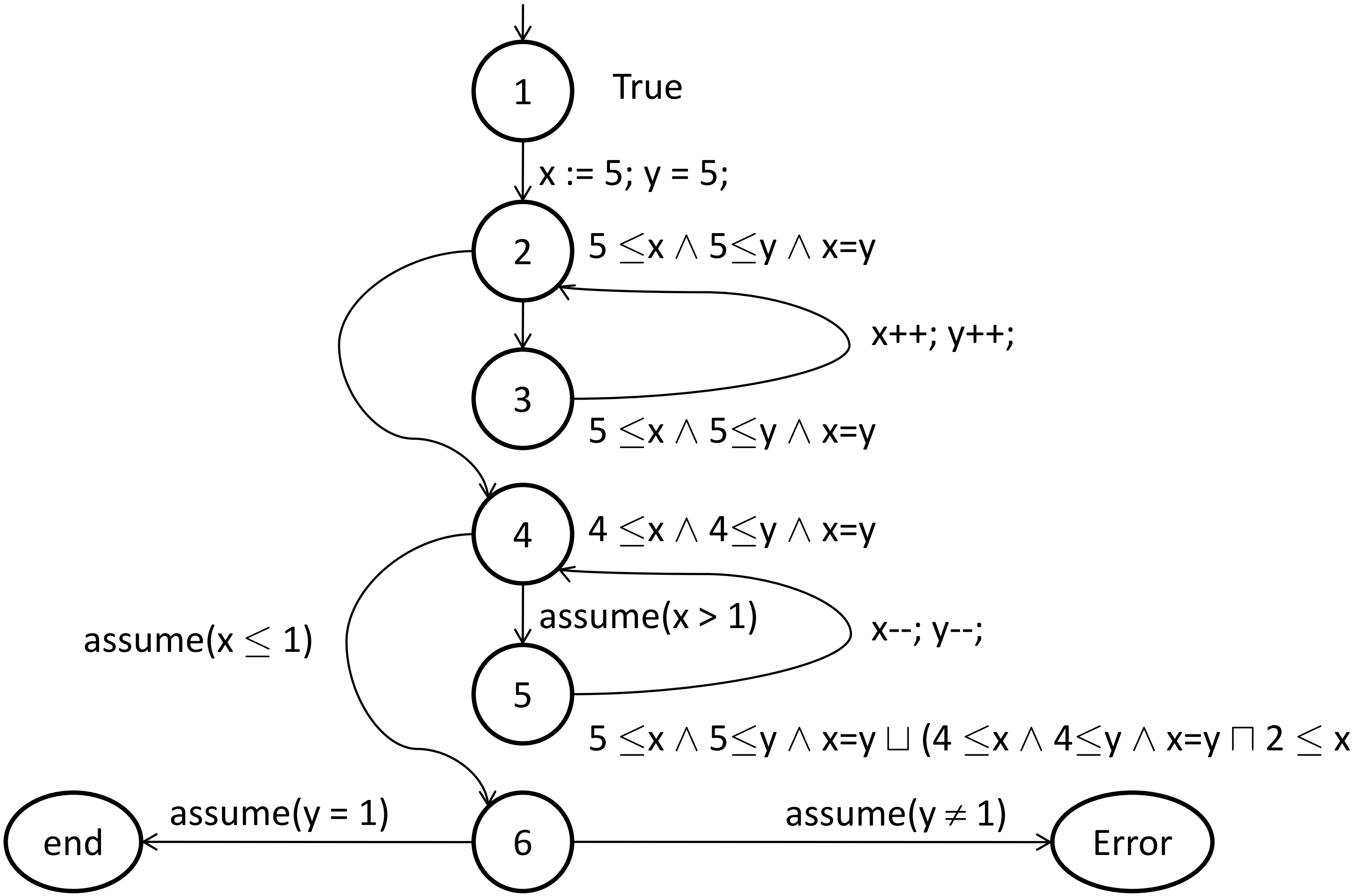
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
5 ≤x ∧ 5≤y ∧ x=y ⊔ 4 ≤x ∧ 4≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Join



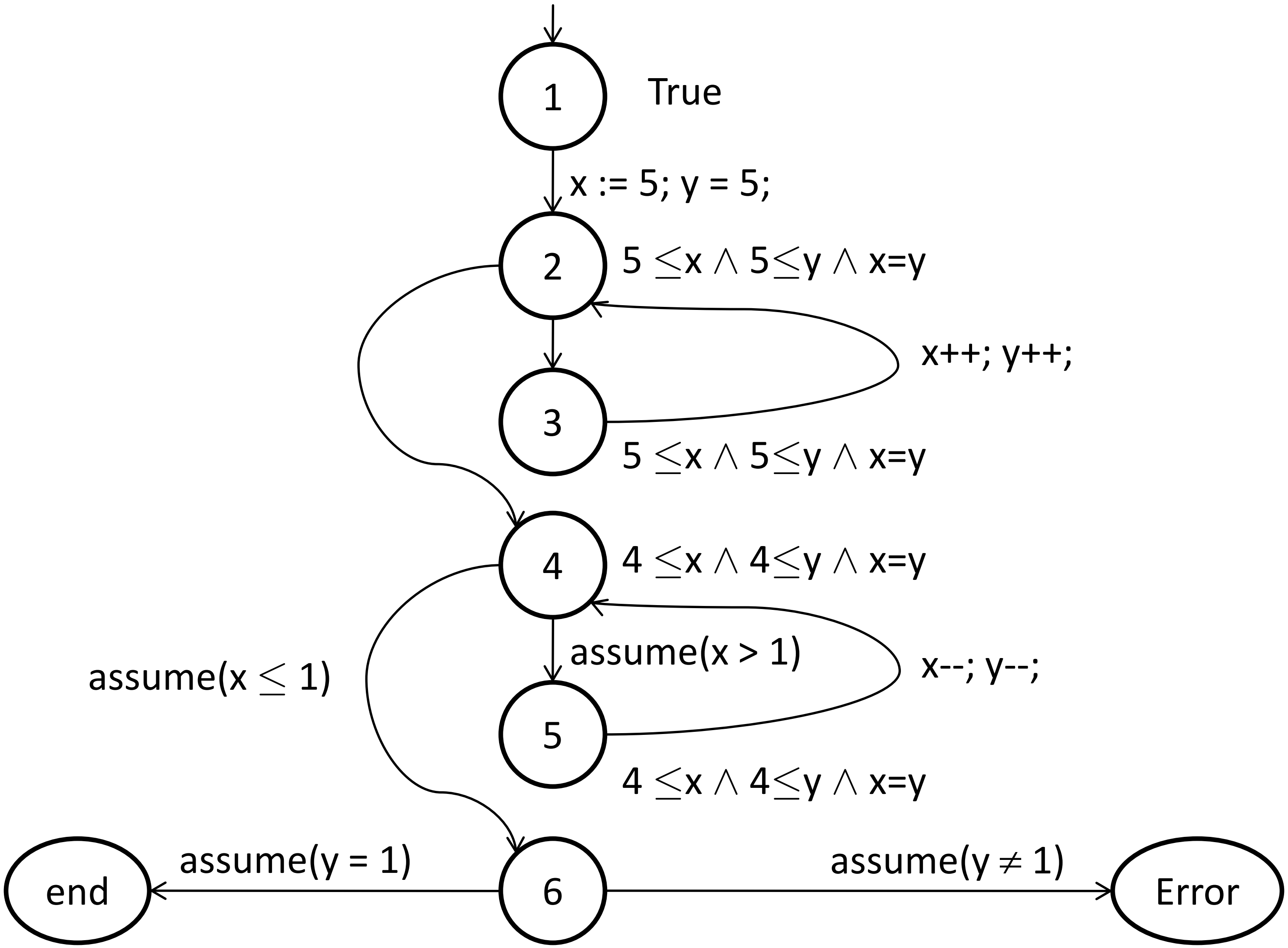
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
4 ≤x ∧ 4≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$post^{\#}$



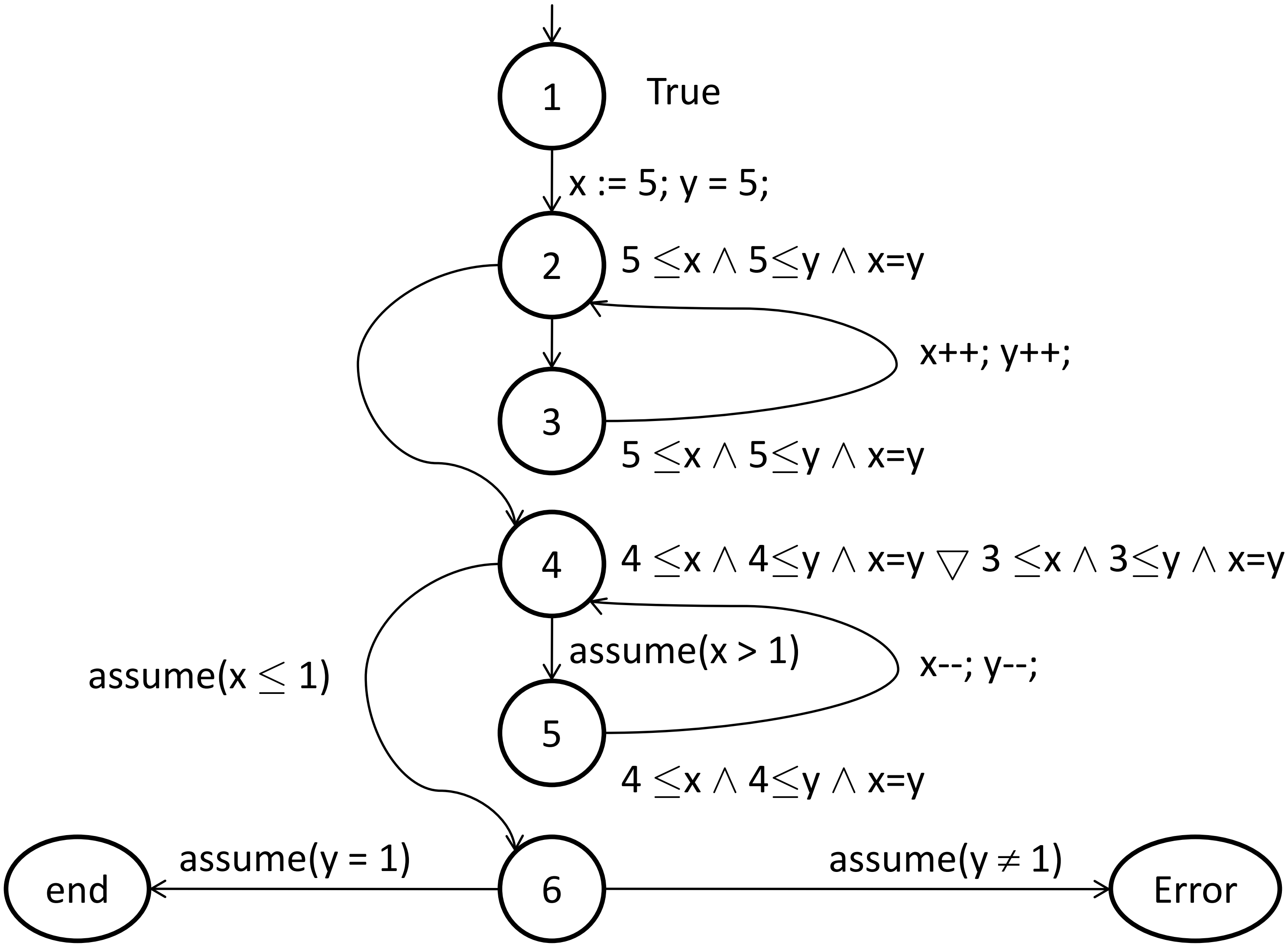
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
4 ≤x ∧ 4≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
5 ≤x ∧ 5≤y ∧ x=y ⊔ (4 ≤x ∧ 4≤y ∧ x=y ⊓ 2 ≤ x)
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Meet



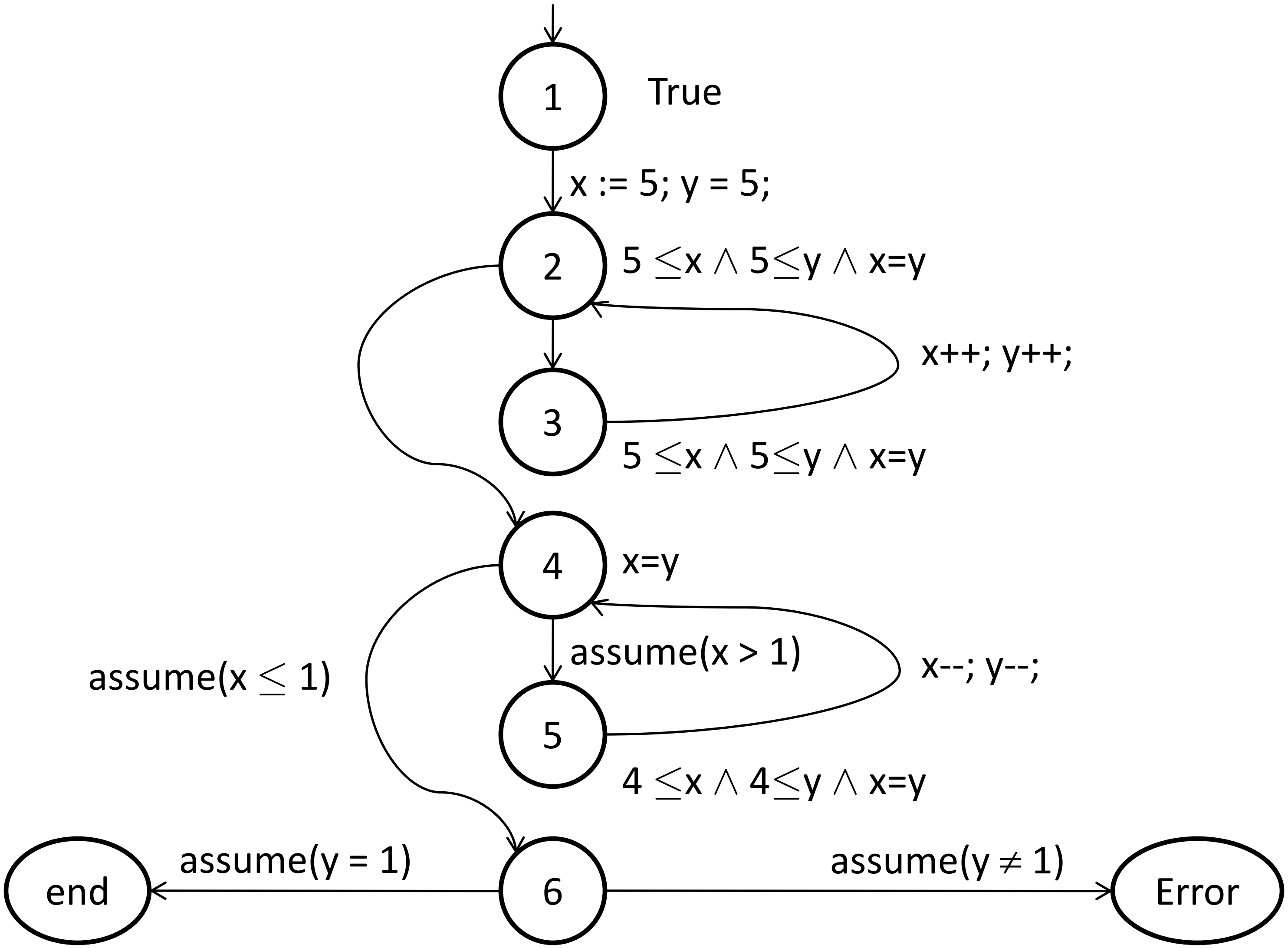
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
4 ≤x ∧ 4≤y ∧ x=y
assume(x ≤ 1)
assume(x > 1)
x--; y--;
5
4 ≤x ∧ 4≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$\text{post}^{\#}$



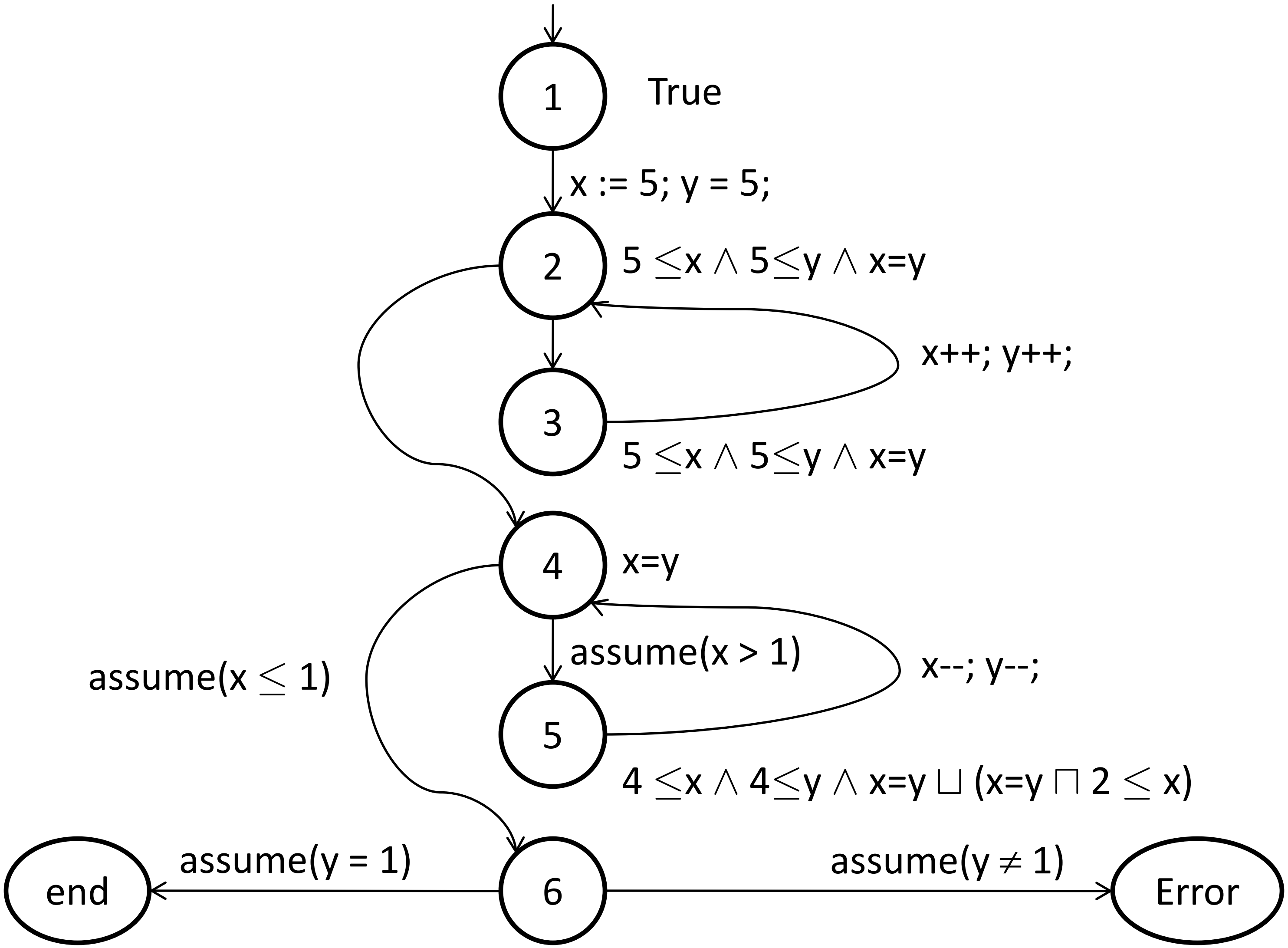
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
4 ≤x ∧ 4≤y ∧ x=y ▽ 3 ≤x ∧ 3≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
4 ≤x ∧ 4≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Widening



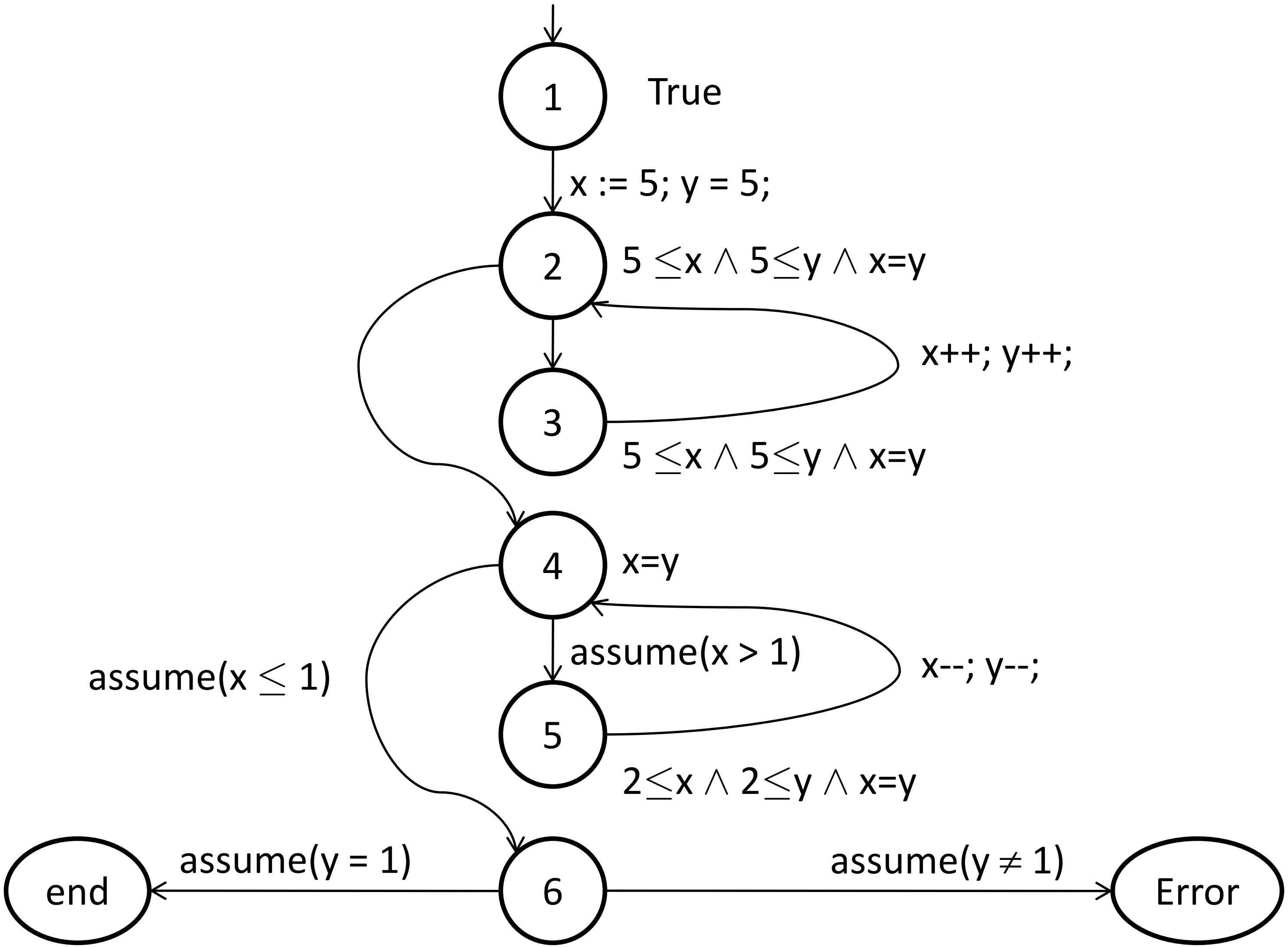
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
4 ≤x ∧ 4≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$\mathbf{post}^\#$

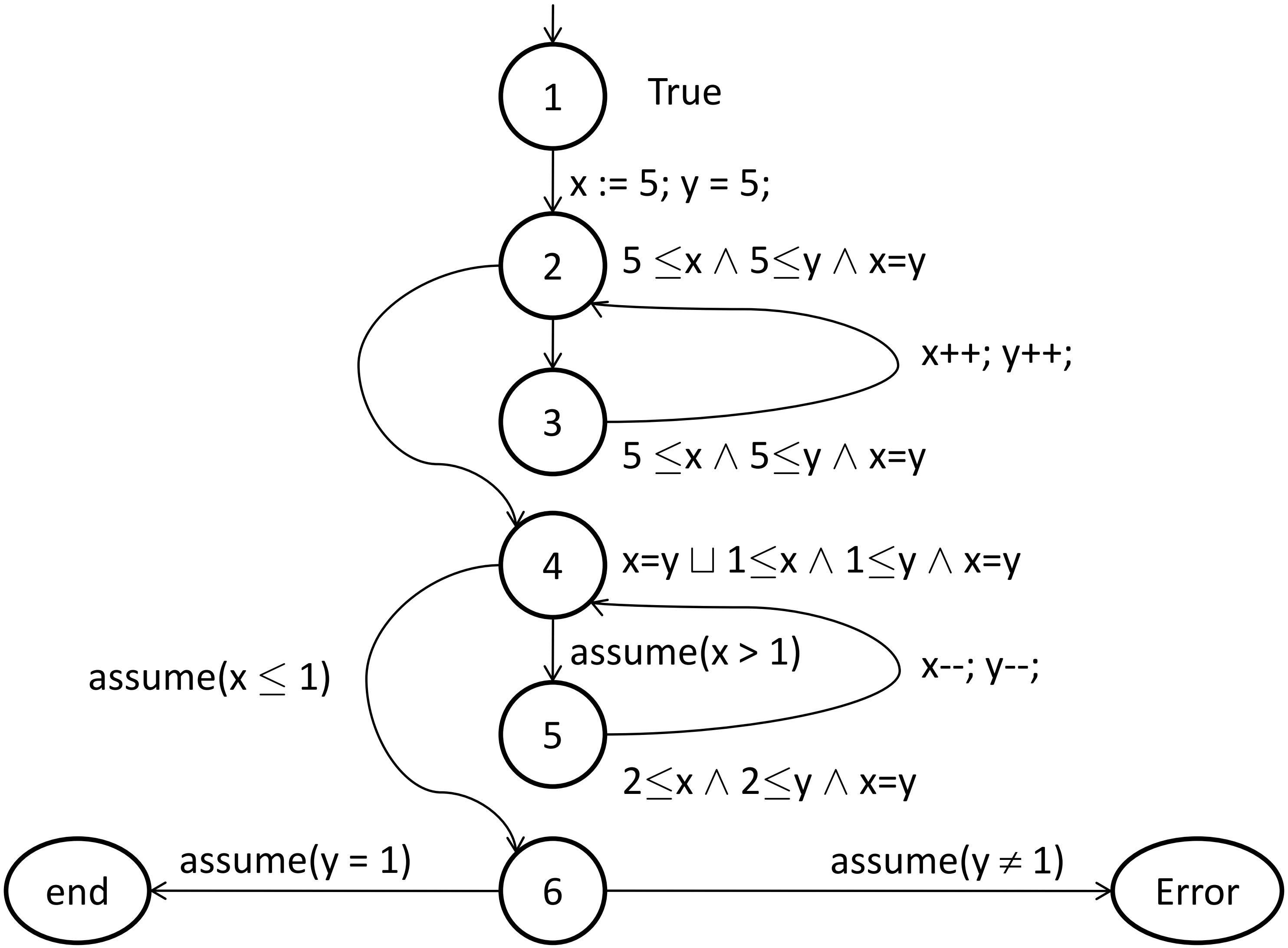


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
4 ≤x ∧ 4≤y ∧ x=y ⊔ (x=y ⊓ 2 ≤ x)
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Meet

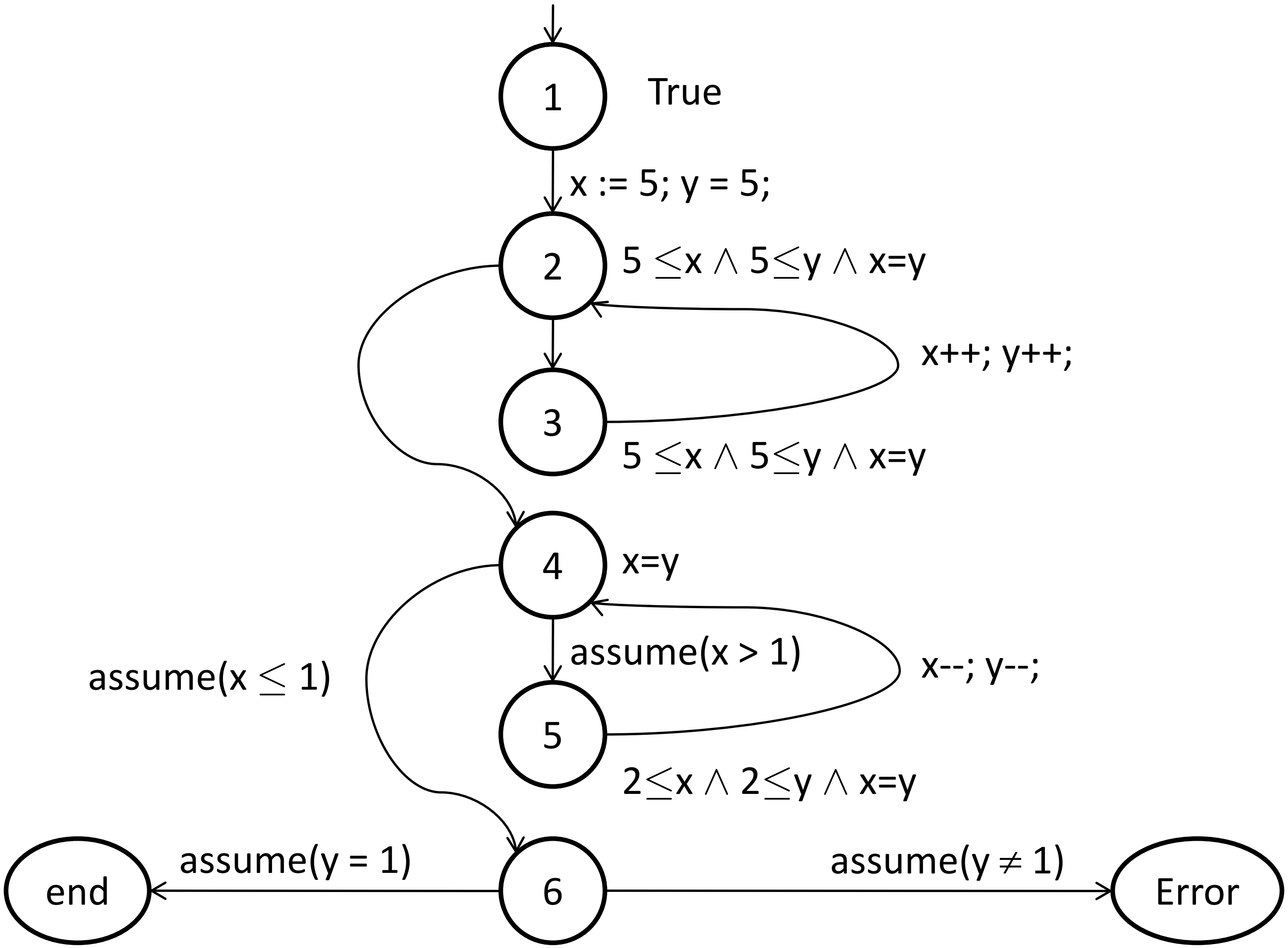

1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$\text{post}^{\#}$

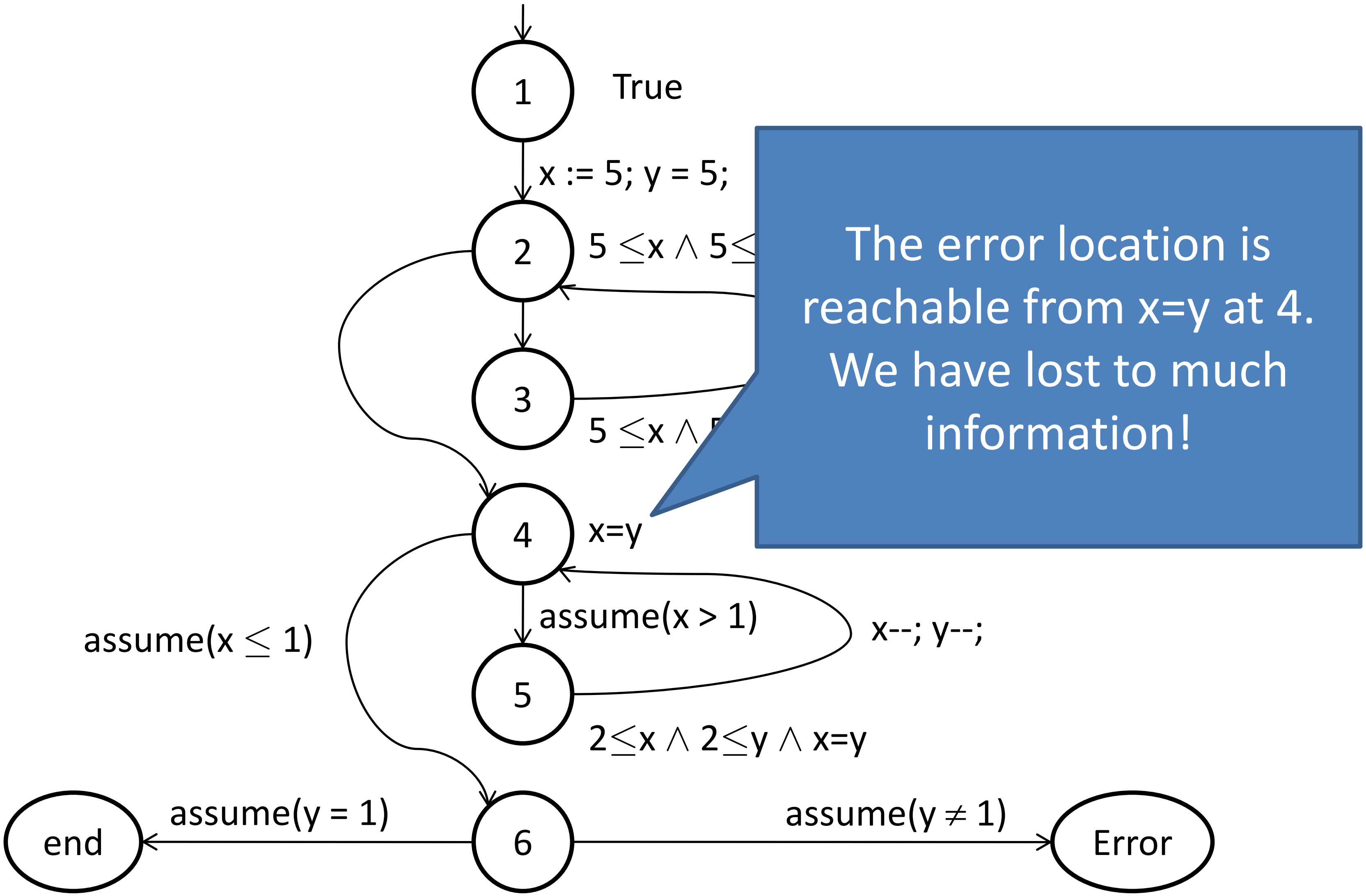


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
x=y ⊔ 1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Join


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

Narrowing



1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤
3
5 ≤x ∧ 5
4
x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error
The error location is reachable from x=y at 4. We have lost to much information!

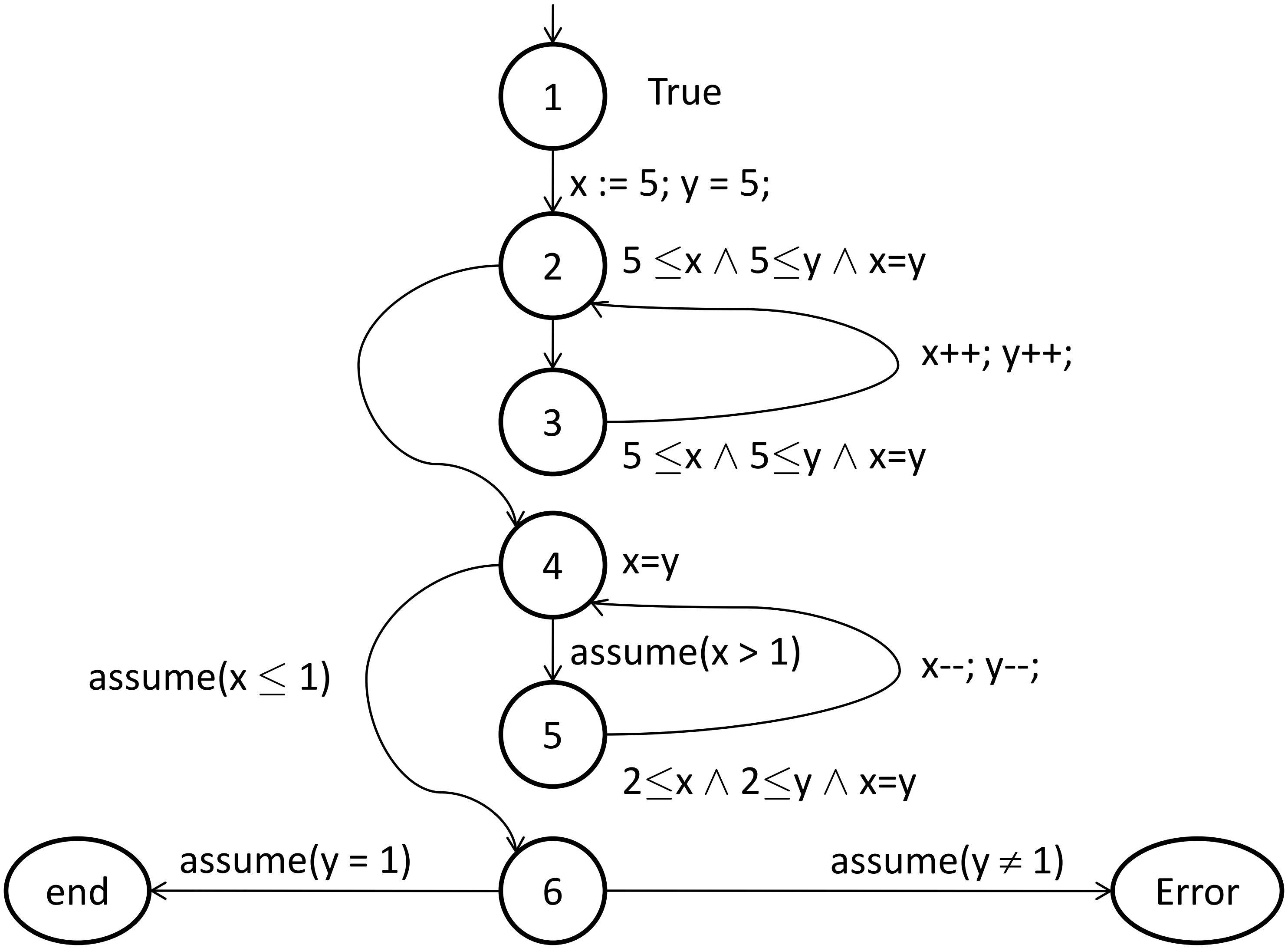
Narrowing

In order to restore some information after widening we apply narrowing!

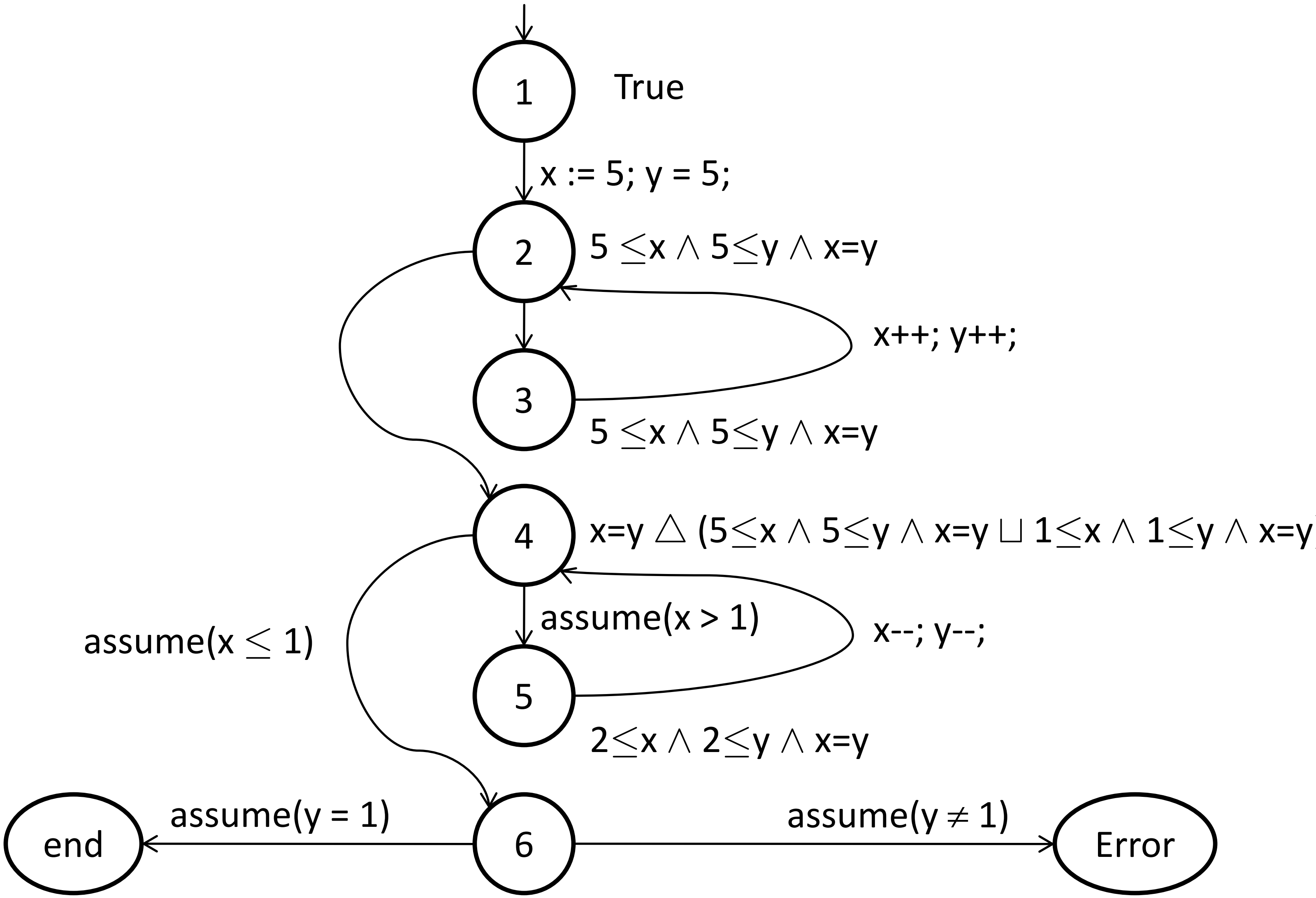
Idea:

Do one more iteration. Keep the inequalities which were not present in the stabilized Zone!

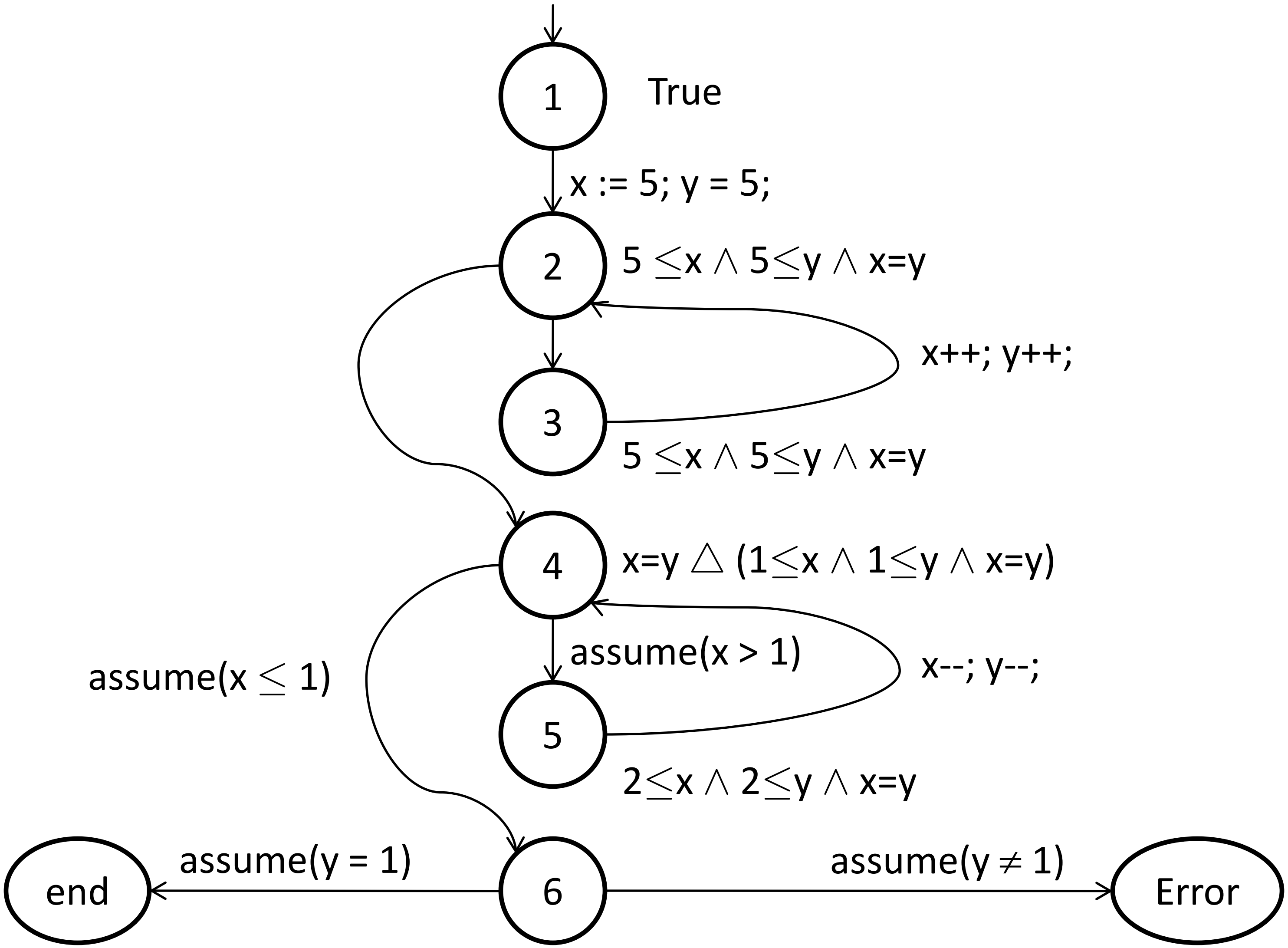
One more iteration



One more iteration



One more iteration



Narrowing

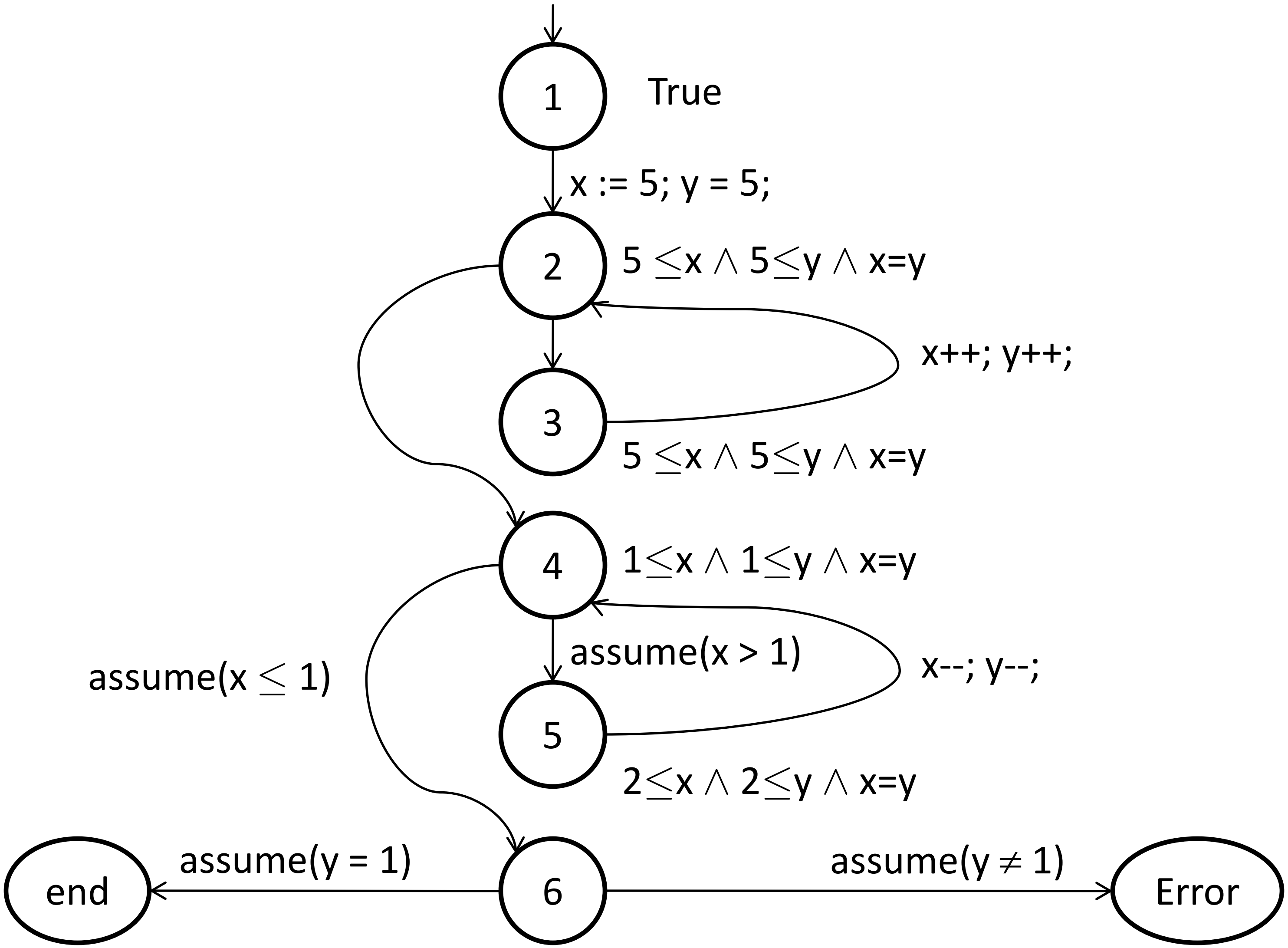
Idea:

Do one more iteration. Keep the inequalities which were not present in the stabilized Zone!

We see, that for x and y no lower bound was part of the stabilized Zone, therefore we keep $1 \leq x$ and $1 \leq y$!

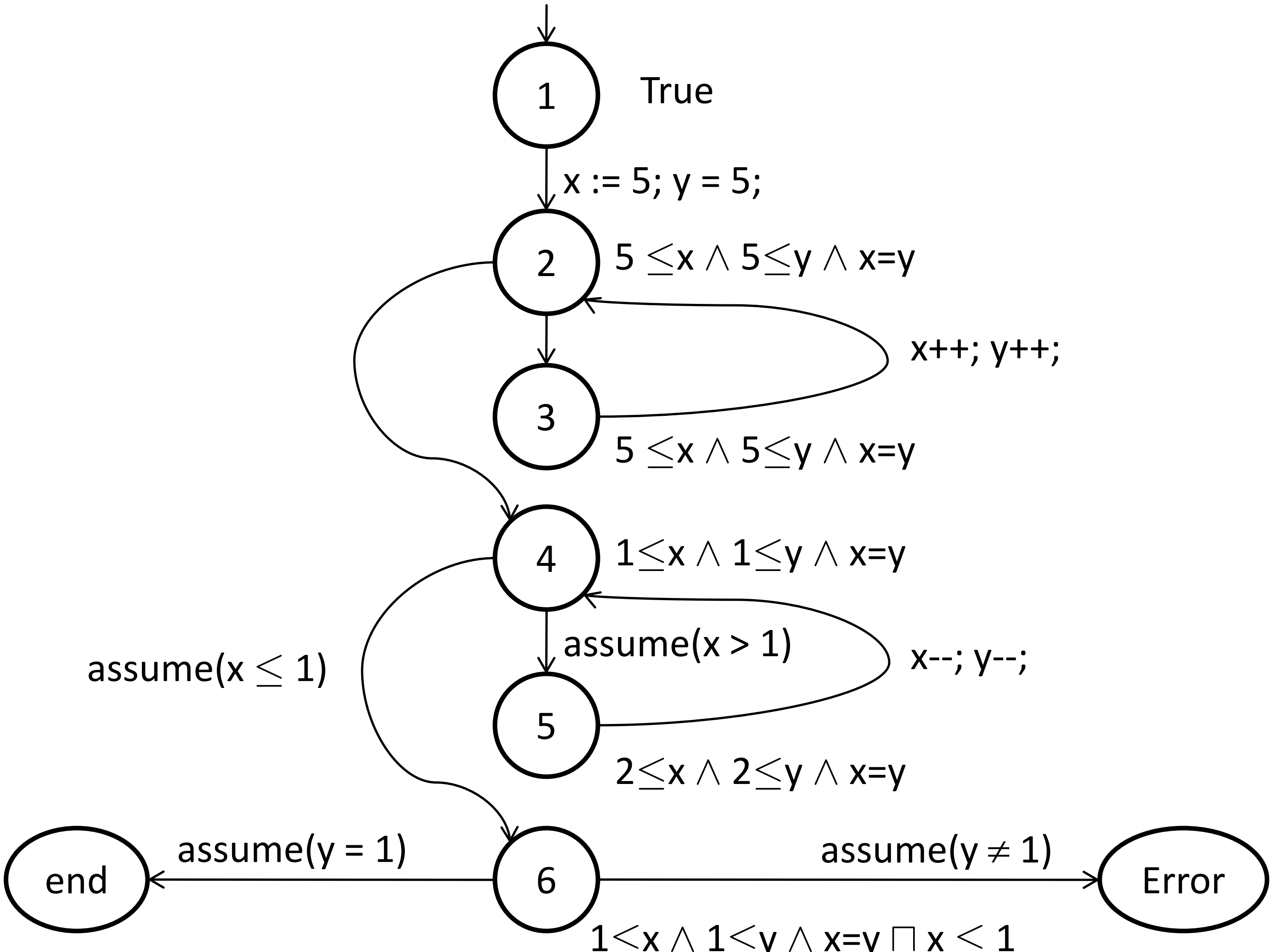
$x=y \triangle (1 \leq x \wedge 1 \leq y \wedge x=y) = 1 \leq x \wedge 1 \leq y \wedge x=y$

Narrowing

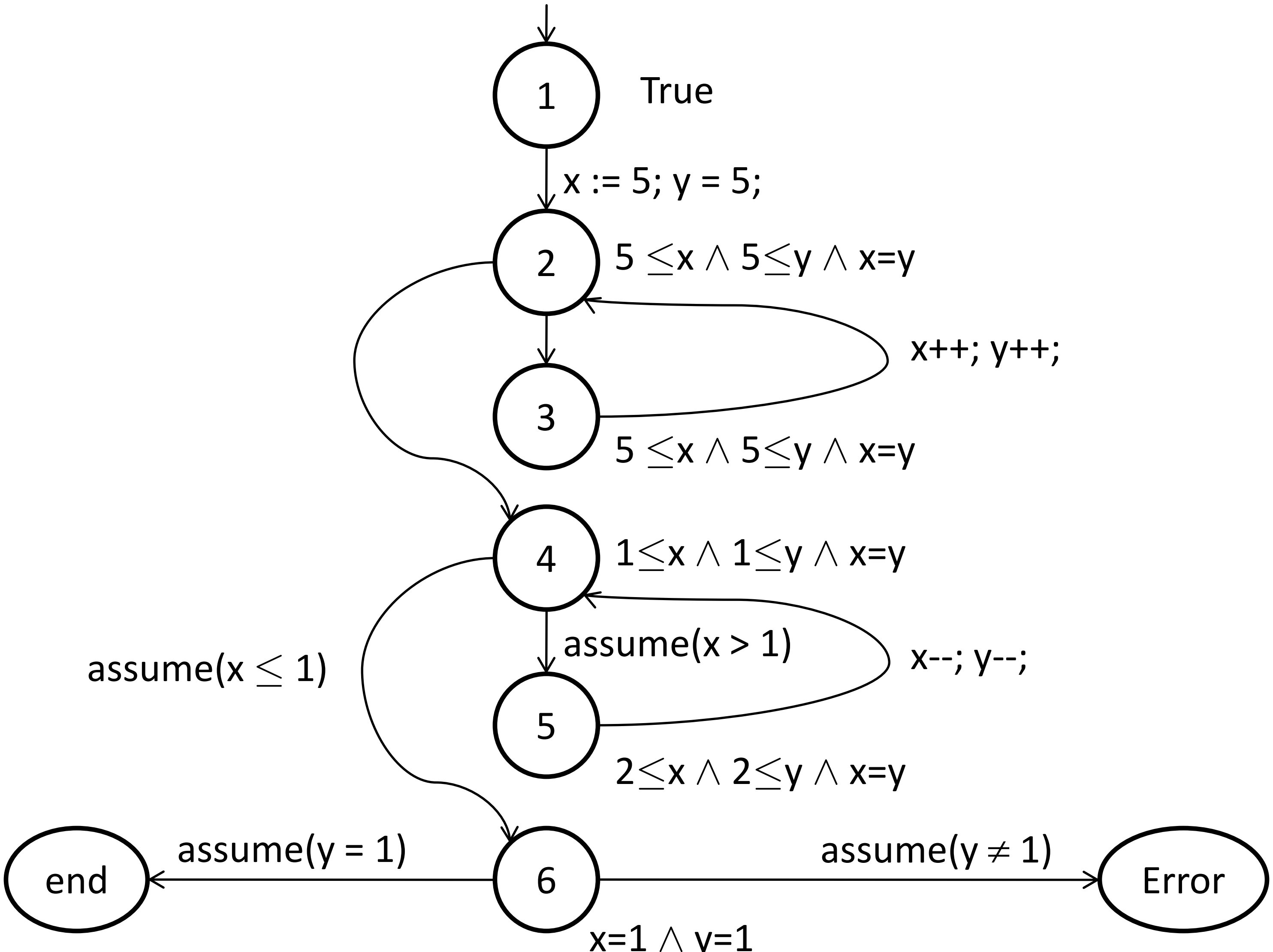


1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error

$post^{\#}$

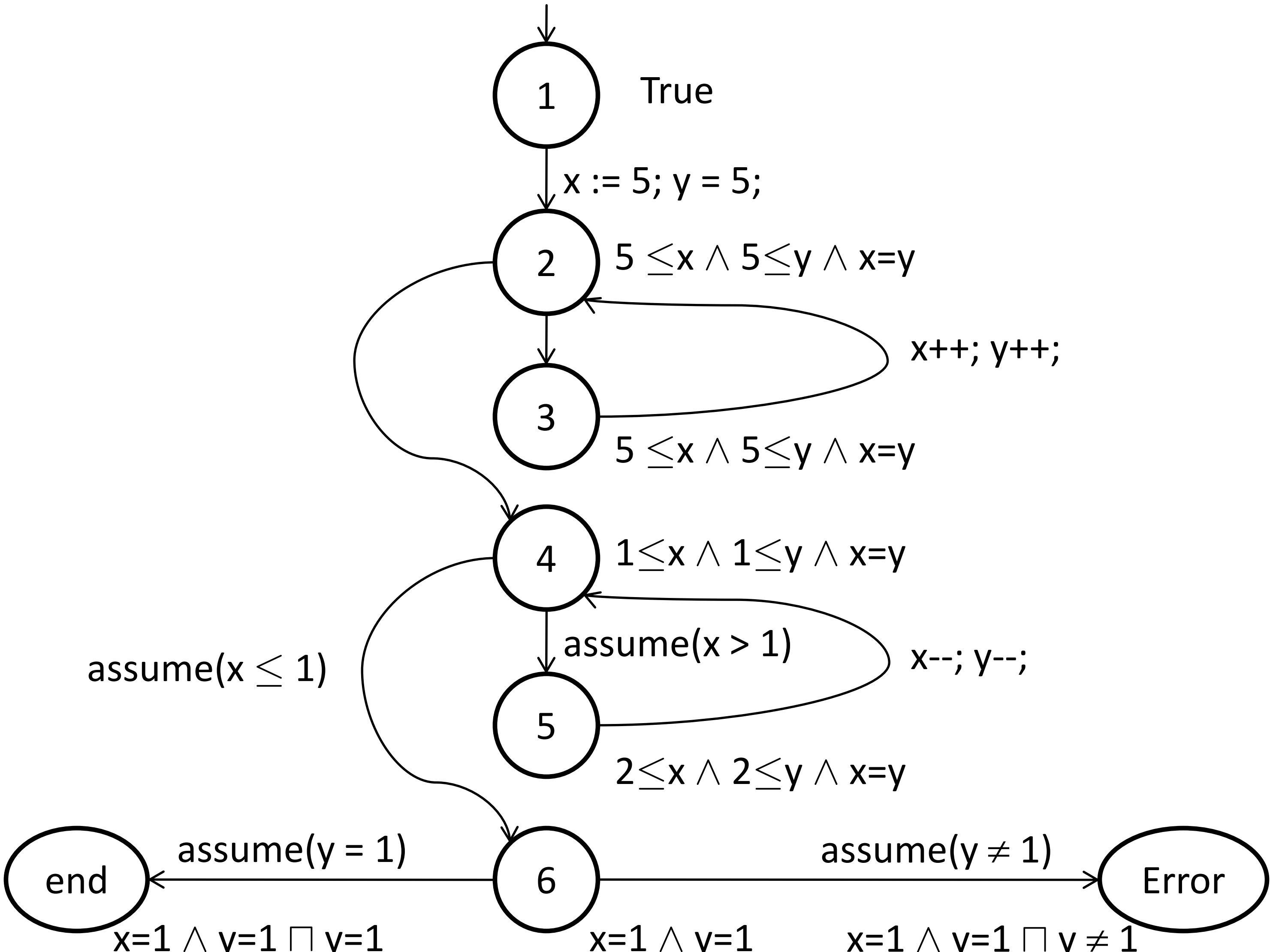

1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error
1≤x ∧ 1≤y ∧ x=y ⊓ x ≤ 1

Meet



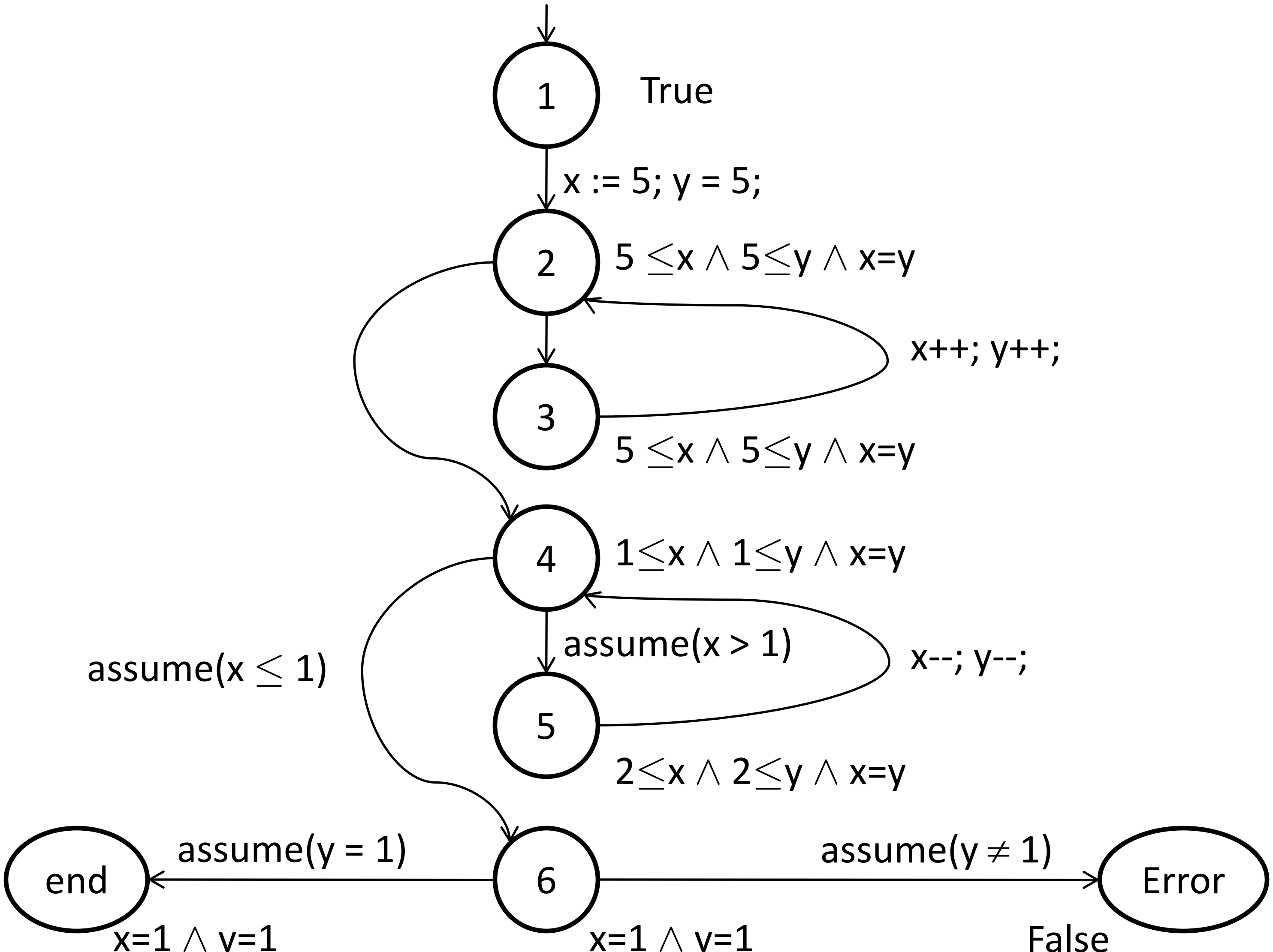
1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error
x=1 ∧ y=1

$post^{\#}$



1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error
x=1 ∧ y=1 ⊓ y=1
x=1 ∧ y=1
x=1 ∧ y=1 ⊓ y ≠ 1

Meet



1
True
x := 5; y = 5;
2
5 ≤x ∧ 5≤y ∧ x=y
x++; y++;
3
5 ≤x ∧ 5≤y ∧ x=y
4
1≤x ∧ 1≤y ∧ x=y
assume(x > 1)
x--; y--;
assume(x ≤ 1)
5
2≤x ∧ 2≤y ∧ x=y
end
assume(y = 1)
6
assume(y ≠ 1)
Error
x=1 ∧ y=1
x=1 ∧ y=1
False

Notes on the Iteration Strategy

- The fixed point computation on abstract elements (as in the above example) can be implemented by a **worklist algorithm**
- The worklist algorithm maintains a list of CFG edges along which abstract elements have to be propagated
- Heuristics can be used to accelerate the fixed-point computation, e.g., stabilizing the current loop before continuing the propagation

Precision of Abstract Interpretation

Join / meet operator normally give the best approximation of union / intersection with regard to the abstract domain.

Attention: Widening/ narrowing give no guarantee about the achieved precision.

A successfull design of widening/ narrowing is based on a good understanding of the application domain.

Outline

1. Introduction
2. Zone Abstract Domain
3. Correctness of Abstract Interpretation
4. Overview on Abstract Domains
5. Discussion of Abstract Interpretation

Correctness of Abstract Interpretation

- How can we know that our analysis is correct?
- Instead of ad-hoc approaches is there a systematic way for giving correctness proofs?
- Can we compose correctness proofs?

$\rightarrow$ simplification of correctness proofs

$\rightarrow$ reusable proof components

Concrete Semantics

A value σ is a mapping from the program variables to the integers, i.e., $\sigma \in Val = Vars \to \mathbb{Z}$.

<u>Goal:</u> For every location $\ell \in Loc$ we want to define the set of values Reach(ℓ) $\in \mathbf{2}^{Val}$ that can reach this location.

Concrete Transfer Function

We use a concrete **transfer function** $\mathbf{post}[\![\cdot]\!]$: $\mathbf{Stmt} \rightarrow \mathbf{2}^{Val} \rightarrow \mathbf{2}^{Val}$ to define the semantics of a transition:

Let (ℓ‘,stmt,ℓ)$\in$Edges be an edge of the labelled transition System and let S $\in \mathbf{2}^{Val}$ be a set of values at ℓ‘:

Then $\mathbf{post}[\![$stmt$]\!]$(S) is the set of values at ℓ after taking this transition.

Concrete Evaluation Function

In order to define $\mathbf{post}[\![\text{stmt}]\!]$ we need a concrete evaluation function $\mathbf{eval}$: $\mathbf{Expr} \to Val \to Val$.

- $\mathbf{eval}$ is defined as expected:
  - $\mathbf{eval}[\![\text{id}]\!](\sigma) = \sigma(\text{id})$
  - $\mathbf{eval}[\![\text{intconst}]\!](\sigma) = \text{intconst}$
  - $\mathbf{eval}[\![E_1 \text{ op } E_2]\!](\sigma) = \text{op}(\mathbf{eval}[\![E_1]\!](\sigma),\mathbf{eval}[\![E_2]\!](\sigma))$

Concrete Transfer Function

- $\mathbf{post}[\![\text{id} = \text{E}]\!](\text{S}) = \{\text{s}[\text{id} \mapsto \mathbf{eval}[\![\text{E}]\!](\text{s})] \mid \text{s} \in \text{S}\}$,
  where $\text{s}[\text{x} \mapsto \text{a}](\text{y}) = \text{s}(\text{y})$, for $\text{x} \neq \text{y}$
  $\text{s}[\text{x} \mapsto \text{a}](\text{y}) = \text{a}$, for $\text{x} = \text{y}$

- $\mathbf{post}[\![\text{assume(E)}]\!](\text{S}) = \{\text{s} \in \text{S} \mid \mathbf{eval}[\![\text{E}]\!](\text{s}) \neq 0\}$,
  where we model *true* as not equal to zero.

Reachable States

The reachable values at every control location

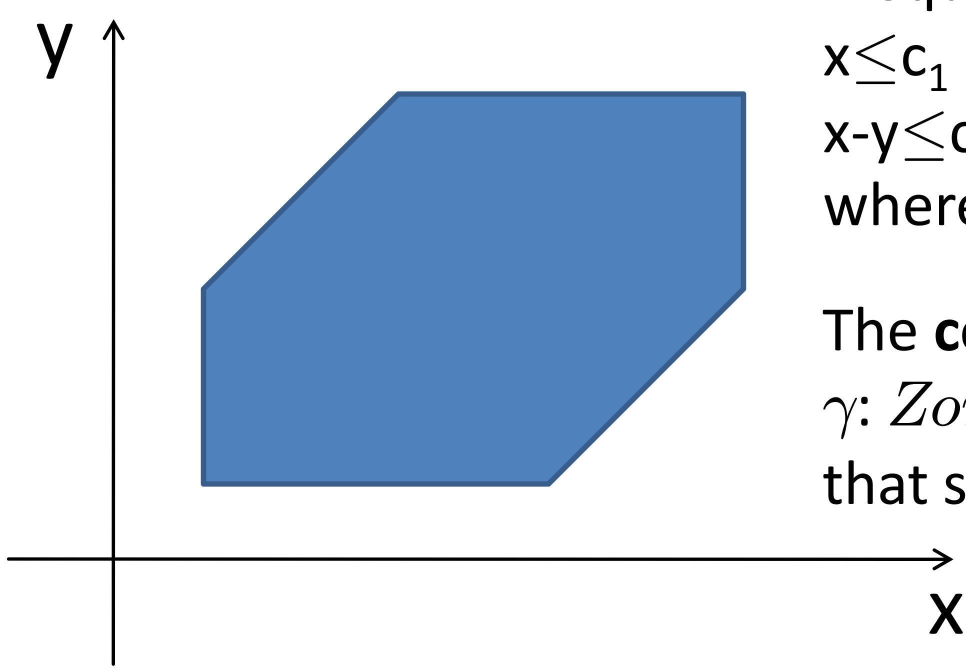
Reach: $Loc \rightarrow \mathbf{2}^{Val}$

are given as the least fixed point of the equation

Reach(ℓ) = $\bigcup_{(\ell',\text{stmt},\ell)\in\text{Edges}} \mathbf{post}[\![\text{stmt}]\!](\text{Reach}(\ell'))$.

The Zone Abstract Domain

The zone abstract domain describes sets of values through certain polygons as pictured below.



Formally, zones consist of linear inequalities:
$x \leq c_1 \wedge -x \leq c_2 \wedge y \leq c_3 \wedge -y \leq c_4 \wedge x-y \leq c_5 \wedge y-x \leq c_6$,
where $c_1,\ldots,c_6 \in \mathbb{Z} \cup \{\infty\}$

The **concretization function** $\gamma: Zone \rightarrow \mathbf{2}^{Val}$ returns all values that satisfy the inequalities.

Partial Order on Zones

Zone $\sqsubseteq$ Zone:

$x \leq c_1 \wedge -x \leq c_2 \wedge y \leq c_3 \wedge -y \leq c_4 \wedge x+y \leq c_5 \wedge x-y \leq c_5 \wedge y-x \leq c_6 \sqsubseteq$

$x \leq d_1 \wedge -x \leq d_2 \wedge y \leq d_3 \wedge -y \leq d_4 \wedge x+y \leq d_5 \wedge x-y \leq d_5 \wedge y-x \leq d_6 \Leftrightarrow$

$c_1 \leq d_1 \wedge c_2 \leq d_2 \wedge c_3 \leq d_3 \wedge c_4 \leq d_4 \wedge c_5 \leq d_5 \wedge c_6 \leq d_6$

The Abstraction Function

We define a **(best-) abstraction function** α: $\mathbf{2}^{Val} \to Zone$ as follows:

α(S) = $\sqcap$\{ a $\in$ $Zone$ | S$\subseteq$$\gamma$(a) \} for all S $\in$ $\mathbf{2}^{Val}$.

Minimum with regard to $\sqsubseteq$ (the order on zones).

Soundness of the Abstraction

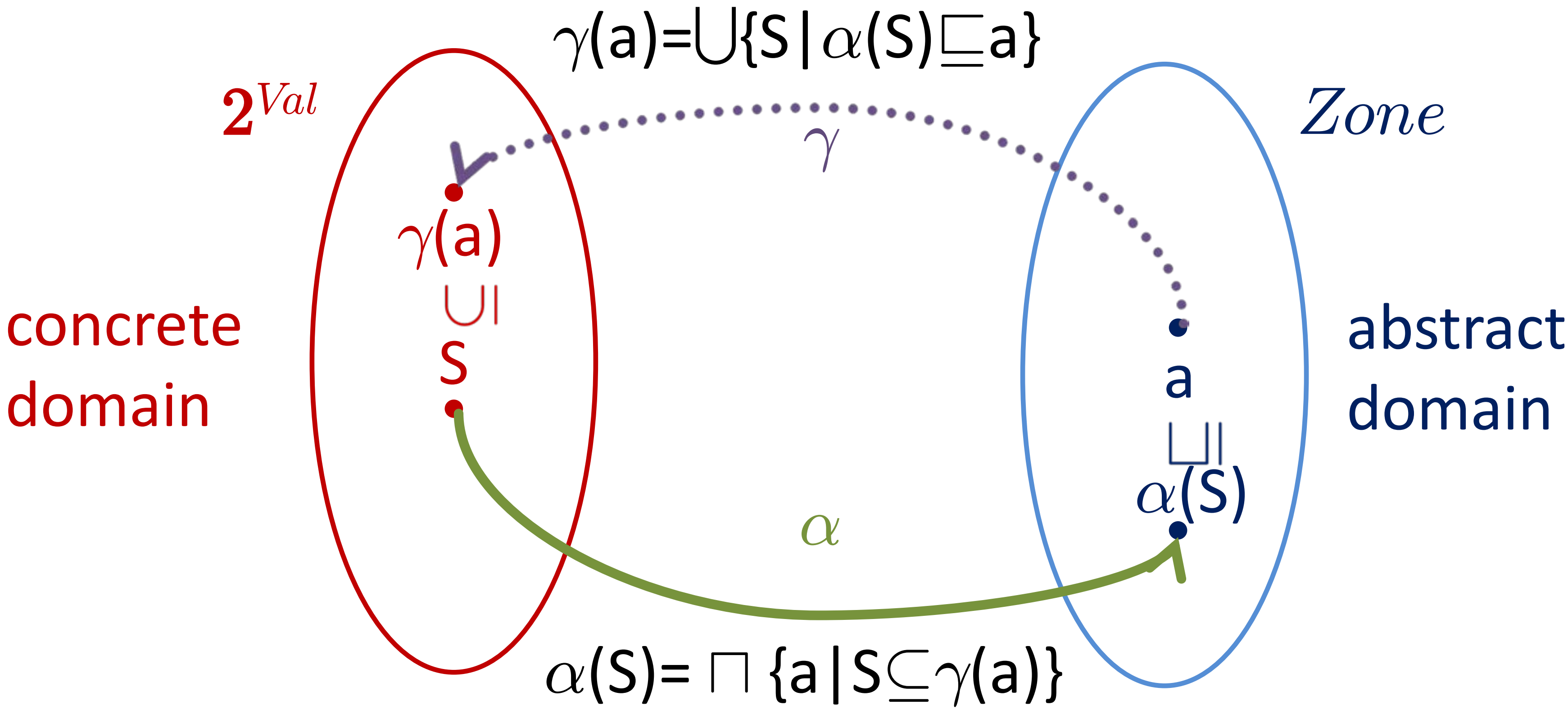
Is the definition of α **sound**, i.e., $\gamma(\alpha(S)) \supseteq S$?

The definition is sound, because γ **preserves arbitrary meets**:

$\gamma(\alpha(S)) = \gamma(\sqcap \{a \in Zone \mid S \subseteq \gamma(a)\}) =$
$\bigcap \{\gamma(a) \mid S \subseteq \gamma(a)\}) \supseteq S$

Because of the above property of γ, the tuple $(Zone, \alpha, \gamma, 2^{Val})$ is a **Galois connection**.

Galois Connection



α is the **lower adjoint**, and
γ is the **upper adjoint**.

Result of Abstract Interpretation with Zones

Abstract interpretation with zones computes a mapping from control locations to zones

Reach$^{\#}$: $Loc \to Zone$

that is a (least) fixed point of the equation

Reach$^{\#}(\ell) = \bigsqcup_{(\ell',\text{stmt},\ell) \in \text{Edges}} \mathbf{post}^{\#}[\![\text{stmt}]\!](\text{Reach}^{\#}(\ell'))$.

Soundness Criterion

Soundness:

For every location ℓ we have Reach(ℓ) $\subseteq \gamma$(Reach$^{\#}$(ℓ)).

Criterion:

For all statments stmt $\in Stmt$, we have

$\mathbf{post}^{\#}[\![\text{stmt}]\!] \sqsupseteq \alpha \circ \mathbf{post}[\![\text{stmt}]\!] \circ \gamma$

Note: $\mathbf{post}^{\#}$ is the *best abstract transformer* for a statment stmt $\in Stmt$,
if $\mathbf{post}^{\#}[\![\text{stmt}]\!] = \alpha \circ \mathbf{post}[\![\text{stmt}]\!] \circ \gamma$.

Outline

1. Introduction
2. Zone Abstract Domain
3. Correctness of Abstract Interpretation
4. Overview on Abstract Domains
5. Discussion of Abstract Interpretation

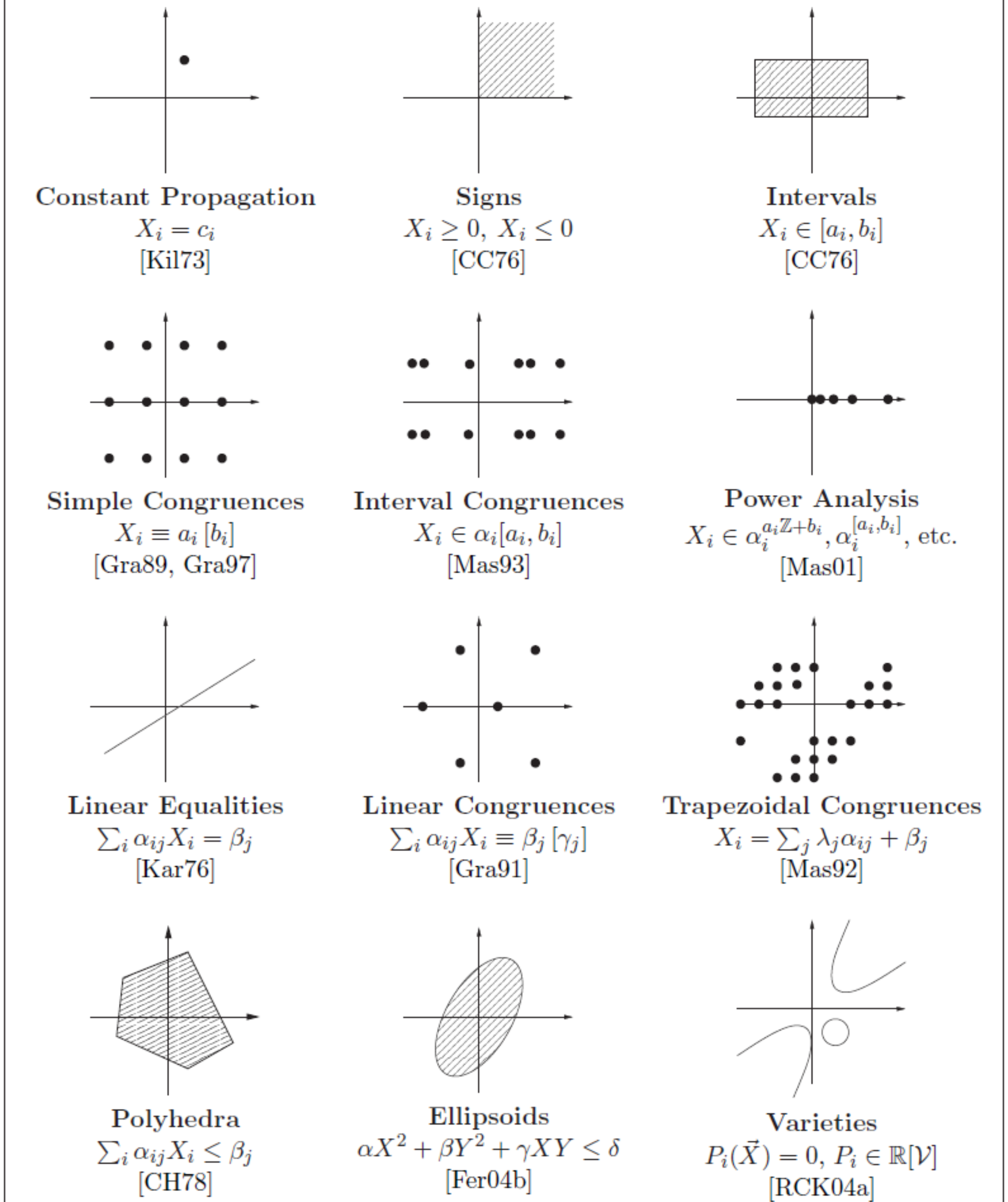
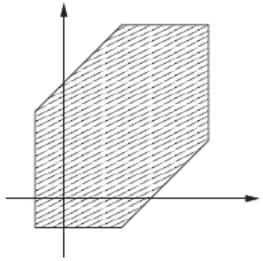
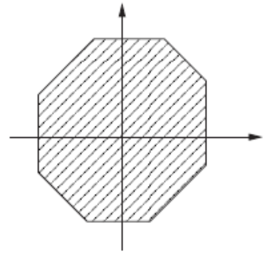


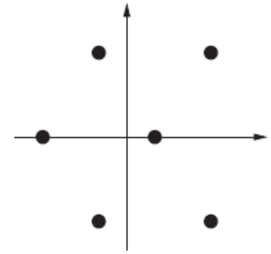
Figure 2.9: The "zoo" of existing numerical abstract domains. The first two rows present non-relational domains and the last two rows present relational domains.



Zones
$X_i - X_j \leq c_{ij}$
[Min01a]

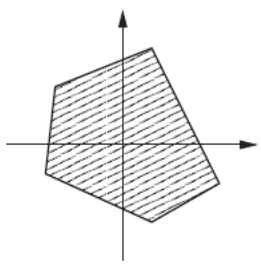


Octagons
$\pm X_i \pm X_j \leq c_{ij}$
[Min01b]

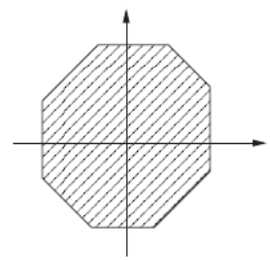


Zone Congruences
$X_i \equiv X_j + \alpha_{ij}\ [\beta_{ij}]$
[Min02]

Figure 2.10: Numerical abstract domains introduced in this thesis.



TVPLI
$\alpha_{ij} X_i + \beta_{ij} X_j \leq c_{ij}$
[SKH02]



Octahedra
$\sum_i \epsilon_{ij} X_i \leq \beta_j,\ \epsilon_{ij} \in \{-1, 0, 1\},\ X_i \geq 0$
[CC04]

Figure 2.11: Other recent numerical abstract domains.

Combining Abstract Domains

E.g., Cartesian product of the interval and congruence abstract domain,

information between the abstract domains can be shared via the reduced product construction:

$x \in [0,7] \wedge x \equiv 0 \bmod 4$

can be reduced to

$x \in [0,4] \wedge x \equiv 0 \bmod 4$

Control Flow Analysis

A language is **higher-order** if a procedure may

i. accept procedures as arguments; and/or
ii. yield procedures as return values.

Why is higher-orderness hard?

- Data-flow depends on control-flow
- Control-flow depends on data-flow

$\rightarrow$ Abstract interpretation allows to construct the CFG and perform data-flow analysis simultaneously

Termination/Bound Analysis

Abstract elements do not describe state invariants but state transitions!

E.g., the size-change abstraction uses Boolean combinations of order relations:

$$x > x' \wedge x > y' \vee y > y' \wedge y > x'$$

Termination is decidable for the size-change abstraction!

Pointer/Shape Analysis

- Long-standing problem
- Shape graphs as abstraction for data structures
- E.g. TVLA, forest automata

Outline

1. Introduction
2. Zone Abstract Domain
3. Correctness of Abstract Interpretation
4. Overview on Abstract Domains
5. Discussion of Abstract Interpretation

Discussion of Abstract Interpretation

- Well-defined dedicated abstract domains
- Combinations of abstract domains
- Framework for defining program semantics as well as static analyses
- Precision of abstract domains can be compared by Galois connections (qualitative comparison)
- Soundness proofs of static analyses with regard to program semantics
- Program analyses can be systematically derived from the program semantics